Efficient Processing of Proximity Based Spatial Queries

by

Muhammad Aamir Cheema

THE UNIVERSITY OF
NEW SOUTH WALES

SYDNEY • AUSTRALIA

A THESIS SUBMITTED IN FULFILMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
(November 2011)

Supervisor: Prof. Xuemin Lin
Joint-supervisor: Dr. Wei Wang

PLEASE TYPE

THE UNIVERSITY OF NEW SOUTH WALES
Thesis/Dissertation Sheet

Surname or Family name: **Cheema**

First name: **Muhammad** Other name/s: **Aamir**

Abbreviation for degree as given in the University calendar: **PhD**

School: **School of Computer Science and Engineering** Faculty: **Faculty of Engineering**

Title: **Efficient Processing of Proximity Based Spatial Queries**

Abstract 350 words maximum: (PLEASE TYPE)

Spatial databases play a vital role in many applications such as Geographic Information Systems (GIS), Computer Aided Design (CAD), Very-Large-Scale Integration (VLSI) designs, Multimedia Information System (MMIS), and medicine and biological research. Due to their importance, a large body of work has focused on efficiently computing various spatial queries. A proximity based spatial query computes the results based on the proximity (closeness) between the objects. Range queries, reverse k nearest neighbors (RkNN) queries and k-closest pairs queries are some of the most important and well studied proximity based spatial queries. In this thesis, we provide efficient solutions for these queries under various settings. Below is a brief description of our contributions.

We present several algorithms to answer RkNN queries under different settings. More specifically, we present an approach, called Lazy Updates, to continuously monitor RkNN queries in Euclidean space as well as in spatial networks. Lazy Updates outperforms all of the existing techniques in terms of both the computation cost and the communication cost. We devise another technique, based on a novel concept of influence zone, to efficiently compute both snapshot and continuous RkNN queries. The influence zone based approach outperforms the existing techniques for both the snapshot and the continuous RkNN queries. We also provide efficient solution for probabilistic RNN queries for the case when the underlying data is uncertain.

We are the first to study the efficient monitoring of moving range queries over a set of static data objects in Euclidean space and in spatial networks. We conduct a rigorous theoretical analysis to show the effectiveness of our approach. The theoretical results are verified by an extensive experimental study. The experimental results also demonstrate that the proposed approach is close to optimal.

We are the first to present a unified framework to answer a broad class of top-k pairs queries including the k-closest pairs queries, k-furthest pairs queries and their bichromatic variants. We conduct a rigorous complexity analysis and show that the expected cost of our proposed algorithms is optimal when the scoring function uses at most two attributes.

Originality Statement

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.

Muhammad Aamir Cheema

Signed ....

Date ....

Dedication

I dedicate this thesis to my parents and my wife.

To my parents because they have always been my nearest neighbors and reverse nearest neighbors[1] [KM00]. They have always been so close to me that I found them standing besides me whenever I needed. It is their unconditional love that motivates me to set higher targets.

To my wife because after marrying her I realized that the best approach to handle missing values is love[2]. With her love, she makes me complete. She is always there to celebrate with me during the good times and is always a strong support during the tough times.

[1]In a not-so-academic wording, your nearest neighbor is an object who is closest to you and your reverse nearest neighbor is an object for which you are the closest object.

[2]Handling missing values in a data set is a widely studied research problem. Various techniques have been proposed to handle the missing values to make the data set as complete as possible. However, each approach has its weaknesses.

Preface

This thesis would not have been possible without the help and support of some very important people in my life. First, I express my gratitude to these wonderful people. Then, I provide the details of the publications resulting from this thesis to specifically acknowledge the help and contributions of the co-authors.

Acknowledgements. Firstly, I would like to thank my parents for their unconditional love, support and encouragement and for being with me on each and every step of my life. I am what I am only because of them.

I am deeply indebted to my mentor and supervisor Prof. Xuemin Lin for his visionary guidance and insightful comments. He always makes sure that he is available to listen to me whenever I have some ideas. A teacher is called a spiritual father and he has played this role very successfully by always providing me full support and guidance. He did not only help me in my research but he also helped me in other dimensions of my life. He deserves a special "thanks" for allowing me to conduct research from my home in Pakistan when I needed to be with my parents.

I am also very thankful to my joint-supervisor Dr. Wei Wang especially for teaching me to always aim for perfection. He has always been very kind, helpful and a great source of knowledge for me. I am very grateful to Dr. Haixun Wang for giving me an opportunity to work as an intern at Microsoft Research Asia on a very interesting and impressive project. I am also thankful to Prof. Jianmin Wang for being a wonderful host when I was an intern at Tsinghua University China. I also learned several important things while working with Dr. Ljiljana Brankovic on one of the papers [CBL$^+$10] (especially the way she theoretically analyzed the problem was quite inspiring).

I should take this opportunity to thank my very helping colleagues. I am especially thankful to Dr. Ying Zhang for always being very kind and helpful. During past few years, I knocked his door infinite number of times for help and he always welcomed me with a warm smile. I am also grateful to other colleagues Wenjie Zhang, Yi Luo, Bin Jiang, Mahady Hasan, Haichuan Shang, Chuan Xiao, Gaoping Zhu, Ke Zhu, Zhitao Shen, Xiang Zhao, Weiren Yu, Liming Zhang and Pengjie Ye for always being very friendly and helping. I feel blessed to be a part of this research group of extremely talented and friendly people.

I am also thankful to my sisters and brothers for their moral and emotional support. A special thanks to my brother, Muhammad Umair Cheema, who is also my best friend. Without his presence, the stay in Australia would not haven been as joyful.

Lastly but the most importantly, I am thankful to my lovely wife, Samar, especially for always praying for my success in research. It can be imagined that I worked quite hard to complete my PhD thesis but let me say that I did not work as much as she prayed for me.

Publications resulting from this thesis. Below is a list of publications resulting from this thesis. I am very grateful to all of the people who collaborated with me for these publications. Their comments and suggestions were always very helping and insightful.

1. Muhammad Aamir Cheema, Xuemin Lin, Ying Zhang, Wei Wang, Wenjie Zhang. "Lazy Updates: An Efficient Technique to Continuously Monitoring Reverse kNN", appeared in *Proceedings of the VLDB Endowment* **(PVLDB)** 2009.

2. Muhammad Aamir Cheema, Wenjie Zhang, Xuemin Lin, Ying Zhang, Xuefei Li. "Continuous Reverse k Nearest Neighbors Queries in Euclidean Space and in Spatial Networks", *The International Journal on Very Large Data Bases* **(VLDBJ)** (accepted in April 2011)

3. Muhammad Aamir Cheema, Xuemin Lin, Wenjie Zhang, Ying Zhang. "Influence Zone: Efficiently Processing Reverse k Nearest Neighbors Queries", appeared in *IEEE International Conference on Data Engineering* **(ICDE)** 2011.

4. Muhammad Aamir Cheema, Xuemin Lin, Wenjie Zhang, Ying Zhang. "Efficiently Processing Snapshot and Continuous Reverse k Nearest Neighbors Queries", *The*

International Journal on Very Large Data Bases **(VLDBJ)** (has been conditionally accepted and is currently under revision).

5. Muhammad Aamir Cheema, Xuemin Lin, Wei Wang, Wenjie Zhang, Jian Pei. "Probabilistic Reverse Nearest Neighbor Queries on Uncertain Data", appeared in *IEEE Transactions on Knowledge and Data Engineering* **(TKDE)** 2010.

6. Muhammad Aamir Cheema, Ljiljana Brankovic, Xuemin Lin, Wenjie Zhang, Wei Wang. "Multi-Guarded Safe Zone: An Effective Technique to Monitor Moving Circular Range Queries", appeared in *IEEE International Conference on Data Engineering* **(ICDE)** 2010.

7. Muhammad Aamir Cheema, Ljiljana Brankovic, Xuemin Lin, Wenjie Zhang, Wei Wang. "Continuous Monitoring of Distance Based Range Queries", appeared in *IEEE Transactions on Knowledge and Data Engineering* **(TKDE)** 2011.

8. Muhammad Aamir Cheema, Xuemin Lin, Haixun Wang, Jianmin Wang, Wenjie Zhang. "A Unified Approach for Computing Top-k Pairs in Multidimensional Space", appeared in *IEEE International Conference on Data Engineering* **(ICDE)** 2011.

To specifically acknowledge the contributions of the co-authors towards this thesis, the table below summarizes the relationship between the publications and the chapters of this thesis. It also briefly describes each problem studied in the corresponding chapter and publications. Please see Chapter 1 for a detailed description of the problems studied in each of the chapters.

Chapters	**Publications**	**Problems studied**
3	1,2	Continuous monitoring of RkNN queries
4	3,4	Influence zone based processing of RkNN queries
5	5	Probabilistic RNN queries on uncertain data
6	6,7	Continuous monitoring of moving range queries
7	8	Top-k pairs queries including k-closest pairs queries

Abstract

Spatial databases play a vital role in many applications such as Geographic Information Systems (GIS), Computer Aided Design (CAD), Very-Large-Scale Integration (VLSI) designs, Multimedia Information System (MMIS), and medicine and biological research. Due to their importance, a large body of work has focused on efficiently computing various spatial queries. A proximity based spatial query computes the results based on the proximity (closeness) between the objects. Range queries, reverse k nearest neighbors (RkNN) queries and k-closest pairs queries are some of the most important and well studied proximity based spatial queries. In this thesis, we provide efficient solutions for these queries under various settings. Below is a brief description of our contributions.

We present several algorithms to answer RkNN queries under different settings. More specifically, we present an approach, called Lazy Updates, to continuously monitor RkNN queries in Euclidean space as well as in spatial networks. Lazy Updates outperforms all of the existing techniques in terms of both the computation cost and the communication cost. We devise another technique, based on a novel concept of influence zone, to efficiently compute both snapshot and continuous RkNN queries. The influence zone based approach outperforms the existing techniques for both the snapshot and the continuous RkNN queries. We also provide efficient solution for probabilistic RNN queries for the case when the underlying data is uncertain.

We are the first to study the efficient monitoring of moving range queries over a set of static data objects in Euclidean space and in spatial networks. We conduct a rigorous theoretical analysis to show the effectiveness of our approach. The theoretical results are verified by an extensive experimental study. The experimental results also demonstrate that the proposed approach is close to optimal.

We are the first to present a unified framework to answer a broad class of top-k pairs queries including the k-closest pairs queries, k-furthest pairs queries and their bichromatic variants. We conduct a rigorous complexity analysis and show that the expected cost of our proposed algorithms is optimal when the scoring function uses at most two attributes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A spatial database system can be defined as a database system that offers spatial objects in its data model and query language, and supports spatial objects in its implementation, providing at least spatial indexing and spatial join methods [Güt94]. Spatial databases are also termed as *image, pictorial, geometric* or *geographic* databases. The applications of spatial databases include Geographic Information Systems (GIS), Computer Aided Design (CAD), Very-Large-Scale Integration (VLSI) designs, Multimedia Information System (MMIS), and medicine and biological research.

The spatial objects are composed of one or more points, lines and/or polygons. Fig. 1.1 shows a map from Google Maps (http://maps.google.com) obtained by entering the query "find computer related businesses near University of New South Wales Sydney". It shows different representations of spatial objects such as points, lines and regions. A point may represent a data object for which only its location is important and its extent in space is not important. For example, the balloons labelled A to J point to the locations of computer shops around the university. The lines represent the facilities of moving through space or connections in space (i.e., roads, rivers). A region represents the spatial object for which its spatial extent is also important. A region may consist of disjoint pieces each containing many polygons. In Fig. 1.1, University of New South Wales and Prince of Wales Hospital are represented by regions.

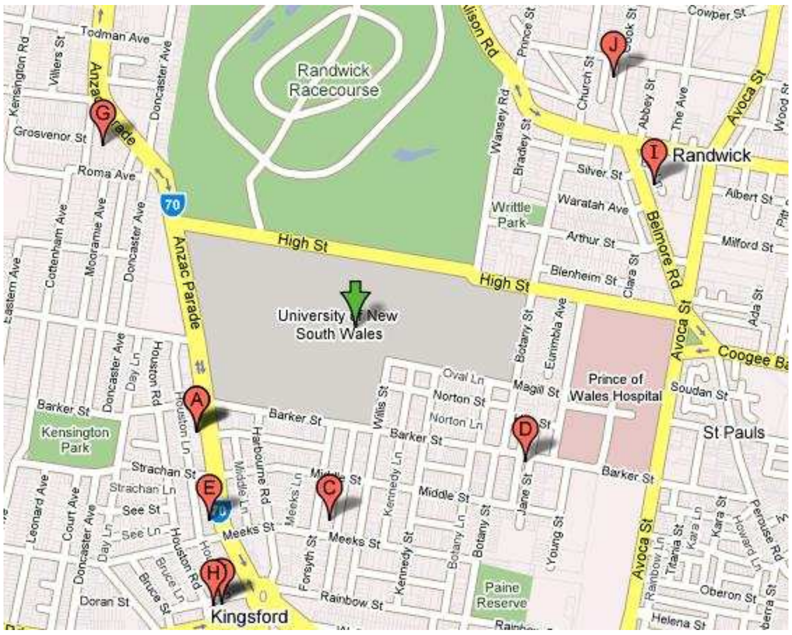


Figure 1.1: Google Maps

Additional functionality must be added in a database system to enable it to process spatial data objects because spatial data objects usually have complex structure and are multidimensional. Moreover, the spatial data objects are usually dynamic and the storage structure should allow efficient insertions and deletions of the objects. A spatial database needs to support different kinds of spatial queries. For example, a query may be issued to find a specific type of spatial objects close to a given spatial object, e.g., find the computer shops near University of New South Wales Sydney. Consider another example where a person may want to find five restaurants nearest to her current location.

To support the search operations on spatial data objects, special data structures are needed to be designed. These data structures are usually called *spatial indexes* or *spatial access methods*. In Section 1.1, we briefly describe why traditional one-dimensional indexes cannot be used. In Section 1.2, we describe few basic spatial queries and some advanced spatial queries. Section 1.3 summarizes the contributions of this thesis towards spatial databases. Thesis organization is presented in Section 1.4.

1.1 A Major Challenge

One of the main challenges in answering the spatial queries is that there is no total ordering among the spatial data objects that preserves spatial proximity. For example, consider that a user wants to find 5 restaurants closest to her location. A straight forward approach is to compute the distances of all the restaurants from her location and report 5 closest restaurants. However, this approach requires accessing all the objects from the spatial database. Another possible approach is to create a one-dimensional index that can be used to answer various queries. For instance, a one-dimensional index can be constructed that contains the distances of all the restaurants from her location sorted in ascending order. To answer her query, we can return first 5 entries from the sorted index. However, this index cannot support a query issued by some other user at a different location. In order to answer the query of this new user, we will have to sort all the restaurants in ascending order of their distances from this user. The difficulty lies in the fact that there is no mapping from multidimensional space into one-dimensional space so that the objects that are close in multidimensional space are also close in the one-dimensional sorted index [GG98].

Another way to answer the queries of above type is to use two sorted lists (i.e., two one-dimensional indexes). First list ($List_x$) contains the restaurants sorted in ascending order of their x-coordinates. The second list ($List_y$) contains them sorted according to their y-coordinates. In order to find 5 closest restaurants from a point p located at (p_x, p_y), we may first find few candidates from $List_x$ that are closest to p_x and then we could calculate their actual distances from p by looking their values in $List_y$. However, this approach can be very inefficient because a restaurant that is closest to p in x-dimension may be the farthest restaurant in y-dimension.

For the reasons mentioned above, traditional one-dimensional indexes such as B-tree [BM72] and extendible hashing [FNPS79] are not suitable for spatial databases. Therefore, in the past two decades, many spatial indexes have been proposed to handle multidimensional spatial data such as *kd*-tree [Ben75], R-tree [Gut84] and its variants (e.g., R*-tree [BKSS90] and R^+-tree [SSH86]). To efficiently answer the spatial queries, most of the existing techniques need to identify interesting problem specific properties and to effectively traverse the existing spatial indexes by exploiting these properties.

1.2 Some Popular Spatial Queries

First, we introduce some basic spatial queries in Section 1.2.1. Then, in Section 1.2.2, we define some popular proximity based spatial queries.

1.2.1 Basic Spatial Queries

Let A and B be two multidimensional spatial data objects (points, lines or regions). Below, we define some basic spatial queries.

- **LENGTH(A).** Return the length of a spatial object (line) A.
- **AREA(A).** Return the area of any two dimensional region A.
- **CENTROID(A).** Return the centroid of a spatial object A. Centroid of a d-dimensional object is the intersection of all hyperplanes that divide it into two parts of equal moment about the hyperplane. Informally, it is the "average" of all points of A.
- **DISTANCE(A, B).** Find the distance between A and B. If A and/or B are not points then the distance function must be defined by the user. For example, a possible definition of the distance between two objects is the minimum distance between them. Other possible definitions include the maximum distance between the objects or the distance between their centroids.
- **EQUALS(A, B).** If A and B have same spatial extent, return true.
- **DISJOINT(A, B).** Return true if A and B are disjoint, i.e., they do not have any point in common.

- **INTERSECTS(A, B).** Return true if A and B intersect each other.
- **CONTAINS(A, B).** If the object B is fully contained in A, return true.

Note that, for all of the above queries, the database system does not need to access any other object from the spatial database. These operations involve geometrical computation based on the locations and the extents of the two spatial objects A and B. The queries mentioned above and other similar queries are more related to the field of computational geometry and are not the focus of this thesis.

1.2.2 Proximity Based Spatial Queries

In this section, we briefly describe the queries that are more advanced and require the database system to access other objects in the database. The distances between the objects play a vital role in the results of these queries and, for this reason, we call such queries proximity based spatial queries. In order to answer these queries, the database system needs to use some special purpose spatial index and query specific properties to reduce the system cost. For such queries, the locations of the spatial objects are important whereas their extents in the space are usually ignored, i.e., the spatial objects are treated as points in the space. Throughout this thesis, we treat spatial objects as points unless mentioned otherwise.

Below, we briefly describe some important proximity based spatial queries.

Range Query

Given a positive value r and a query point q, a range query returns all the objects that are within distance r from q. In other words, the range query returns every object o for which $dist(q, o) \leq r$ where $dist(q, o)$ denotes the distance between q and r.

Consider the example of Fig. 1.2 where a query object q and 4 other objects o_1 to o_4 are shown in a Euclidean space. A range query q with range r returns the objects o_1 and o_2 because these two objects have distances from q at most equal to r.

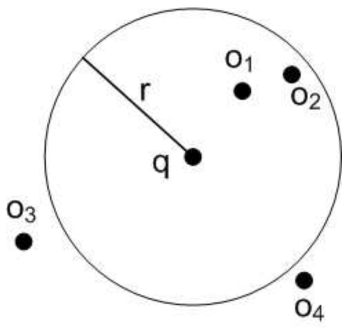


Figure 1.2: Proximity based spatial queries

k Nearest Neighbors (kNN) Query

A nearest neighbor query returns the object that is closest to the query q. A k nearest neighbors (kNN) query returns k objects closest to the query q. Formally, a kNN query returns a set N_k consisting of k objects such that for any object $o \in N_k$ and for any other object $o' \notin N_k$, $dist(q, o) \leq dist(q, o')$.

In the example of Fig. 1.2, the closest object from q is o_1. Hence, a kNN ($k = 1$) query returns o_1 as the answer.

Reverse k Nearest Neighbors (RkNN) Query

A reverse k nearest neighbors (RkNN) query returns every object o such that q is one of the k nearest neighbors of o. When $k = 1$, a RkNN query is denoted as RNN query.

Consider the example of Fig. 1.2, the query object q is the closest object of o_3. Hence, o_3 is a RNN of q. Similarly, o_4 is also a RNN because q is the closest object of o_4. Note that although the object o_1 is the nearest neighbor of q, it is not the reverse nearest neighbor of q because the closest object of o_1 is not q but is o_2. Also note that a RkNN query may return zero or more objects whereas a kNN query returns exactly k objects.

k Closest Pairs Query

A k closest pairs query returns k pairs of objects with the smallest distances between them. Let p be a pair of objects (o, o') and assume that $p.score = dist(o, o')$ be the distance between o and o'. A k closest pairs query returns a set S consisting of k pairs such that for any pair $p \in S$ and for any other pair $p' \notin S$, $p.score \leq p'.score$. In the example of Fig. 1.2, the closest pair is (o_1, o_2).

1.2.3 Various Problem Settings for Spatial Queries

Due to the importance of the spatial queries in a variety of applications, these queries are studied using various problem settings. Below, we briefly describe some common problem settings used for the spatial queries.

Snapshot vs Continuous

In a snapshot query, the results of the query are to be computed only once. For example, a user may want to find restaurants within 5 miles of University of New South Wales. He may issue a snapshot range query with range set to 5 miles and query location set as University of New South Wales.

In contrast to the snapshot queries, a continuous query requires the results to be continuously updated as the underlying data is updated. For instance, a person driving a car may want to find the restaurants within 5 miles of his current location. Since the car is continuously moving, the results are required to be updated continuously. He may issue a continuous range query with range set to 5 miles and the query location set as the location of car. While in the above example only the query is moving, in many applications, all of the query objects and data objects may be continuously moving. For instance, a person driving a taxi might want to continuously monitor other taxis within 5 miles of his location. In this example, the query and the data objects all are continuously moving.

Euclidean Space vs Spatial Networks

Recall that the definitions of the proximity based spatial queries use a distance function $dist()$ which returns the distance between the objects. Depending on the applications, different variations of the distance functions are used such as Euclidean distance, Manhattan distance and network distance etc. For instance, a person driving a car may be interested in nearby restaurants according to the road distance (or according to the time required to reach there). Hence, he may issue a query on the road network such that the distance function $dist(q, o)$ returns the network distance between q and o. Similarly, a fighter pilot may want to find the nearby enemy targets. He may issue a spatial query in Euclidean space where the distance function $dist(q, o)$ returns Euclidean distance between q and o.

Certain Data vs Uncertain Data

Usually, it is assumed that the exact values of objects (e.g., locations) are known and the spatial queries use these locations. However, uncertain data is inherent in many important applications such as sensor databases, moving object databases, market analysis, and quantitative economic research. In these applications, the exact values of data might be unknown due to limitation of measuring equipment, delayed data updates, incompleteness, or data anonymization to preserve privacy. In such applications, the spatial queries are issued on the uncertain data and probabilistic results are returned.

1.3 Contributions

In this section, we summarize our contributions in this thesis. We proposed efficient techniques for several important spatial queries. For each of these queries, we briefly describe our contributions.

1.3.1 Reverse k Nearest Neighbor Queries

In this thesis, we study Reverse k Nearest Neighbors (RkNN) queries under different settings. Below is a summary.

Snapshot RkNN Queries

We present an efficient approach to answer snapshot RkNN queries. Our approach is based on a novel concept of *influence zone* which is an area such that an object o is one of the RkNNs of q if and only if o is inside the influence zone. We present efficient techniques to compute the influence zone. Once the influence zone is computed, the algorithm retrieves the objects inside it and reports the results. We conduct extensive experiments to demonstrate that the proposed approach outperforms all existing techniques in terms of CPU time as well as I/O cost.

This research [CLZZ11] was published in *IEEE International Conference on Data Engineering* **(ICDE)** 2011. In an extended version, we present techniques to efficiently update the influence zone when the underlying data set may change due to insertions of new objects or deletions of the existing objects. An extended version [CLZZ] has been submitted to *The International Journal on Very Large Data Bases* **(VLDBJ)** and is currently under review.

Continuous Monitoring of RkNN Queries

Continuous RkNN Queries in Euclidean Space. We present two techniques namely Lazy Updates [CLZ+09] and InfZone [CLZZ11] to continuously monitor RkNN in Euclidean space. Lazy Updates monitors RkNN queries for the case when all the queries and data objects are continuously moving. The proposed technique assigns each object and query a rectangular region called *safe region*. We develop efficient and effective pruning rules that are used to prune the space based on the safe regions of the objects and the queries. The expensive pruning phase is not required to be called as long as the query and the objects remain in their respective safe regions. This saves the overall computation time. Moreover, the objects are required to report their new locations only when they leave their respective safe regions. This reduces the communication cost of the system. A preliminary version of Lazy Updates [CLZ+09] appeared in *Proceedings of the VLDB Endowment* **(PVLDB)** 2009.

In contrast to Lazy Updates, InfZone aims to answer RkNN queries when only the data objects are moving and the queries do not move. In such problem settings, InfZone performs significantly better than Lazy Updates. InfZone computes the influence zone which is the area such that an object o is one of the RkNNs of q if and only if o is inside the influence zone. To continuously update the results, the algorithm only needs to monitor the objects that leave or enter the influence zone. This research [CLZZ11] was published in *IEEE International Conference on Data Engineering* **(ICDE)** 2011.

Continuous RkNN Queries in Spatial Networks. We extended Lazy Updates to continuously monitor RkNN queries in spatial networks. We present several pruning rules to efficiently prune the search space and propose techniques to answer RkNN queries and its variants in spatial networks. Our approach significantly reduces the processing time as well as the communication cost. This extended version [CZL+11] appeared in *The International Journal on Very Large Data Bases* **(VLDBJ)**.

RNN Queries on Uncertain Data

We present efficient techniques to answer probabilistic RNN queries on uncertain data. A probabilistic RNN query returns every object o such that the probability of o to be the RNN of q is greater than a given threshold. We propose several novel pruning rules to quickly shortlist the candidate objects. Then, we use efficient verification techniques to compute the exact probabilities of these candidate objects to be the RNN of q. Our approach answers RNN queries in multidimensional Euclidean space. Using real and synthetic data sets, we illustrate the efficiency of our proposed approach.

This research [CLW+10] was published in *IEEE Transactions on Knowledge and Data Engineering* **(TKDE)**.

1.3.2 Continuous Monitoring of Moving Range Queries

We study the problem of continuously monitoring moving range queries on a set of data objects that do not change their locations. Consider the example of a person driving a car who is interested in fuel stations within 5 miles. In this example, the query (i.e., car) is continuously moving whereas the objects (i.e., fuel stations) are static. We present techniques to answer the moving range queries in Euclidean space as well as in spatial networks. Below are the details.

Moving Range Queries in Euclidean Space

Our proposed technique is based on the concept of *safe zone*. A safe zone is an area such that as long as the query remains inside the safe zone the results of the query are not required to be updated. We propose efficient techniques to compute and update the safe zone. Our system reduces the overall cost because it does not require to recompute the results as long as the query remains in its safe zone.

We conduct a rigorous theoretical analysis to study the effectiveness of our safe zone based approach. The accuracy of the theoretical analysis is verified by an extensive experimental study. Moreover, the experimental results demonstrate that the proposed approach is close to optimal.

This research [CBL+10] was published in *IEEE International Conference on Data Engineering* **(ICDE)** 2010.

Moving Range Queries in Spatial Network

We extend our safe zone based approach to answer the moving range queries in the spatial network. This extended version [CBL+11] appeared in *Special Issue of IEEE Transactions on Knowledge and Data Engineering* **(TKDE)** *on Best Papers of ICDE 2010.*

1.3.3 k-Closest Pairs Queries

As mentioned earlier, a k-closest pairs query returns k pairs of objects with the smallest distances. We first generalize this problem to a top-k pairs problem. A top-k pairs query returns k pairs with the smallest scores where the score of each pair is computed by using a user specified scoring function. In this research, we present a unified approach to answer a broad class of top-k pairs queries including the k closest pairs queries, the k furthest pairs queries and their variants. We provide a detailed complexity analysis and show that the expected performance of the proposed algorithms is optimal when the scoring functions involve less than three attributes. Extensive experiments demonstrate the efficiency of our proposed algorithms.

This research [CLW+11] was published in *IEEE International Conference on Data Engineering* **(ICDE)** 2011.

1.4 Thesis Organization

This dissertation is organized as follows.

- Chapter 2 provides a survey of the related work.
- Chapters 3, 4 and 5 present our research on reverse k nearest neighbors queries. Below, is a more specific description.
  - Chapter 3 presents our technique, named Lazy Updates [CLZ+09, CZL+11], to continuously monitor RkNN queries in Euclidean space and in spatial networks.
  - Chapter 4 covers our influence zone based techniques [CLZZ11, CLZZ] to answer snapshot and continuous RkNN queries.
  - Chapter 5 presents our algorithm [CLW+10] to answer probabilistic reverse nearest neighbors queries on uncertain data.
- Chapter 6 describes our techniques [CBL+10, CBL+11] to continuously monitor moving range queries in Euclidean space and in spatial networks.
- Chapter 7 presents our unified framework [CLW+11] to answer a broad class of top-k pairs queries including k-closest pairs queries, k-furthest pairs queries and their variants.

- Chapter 8 concludes our research, describes some of the open problems and provides several possible directions for future work.

Chapter 2

Related Work

In this chapter, we provide a brief overview of the related work for each type of queries we studied in this thesis. More specifically, we provide the related work on reverse nearest neighbor queries in Section 2.1 followed by an overview of the related techniques for range queries in Section 2.2. In Section 2.3, we provide a brief description of the existing techniques to answer probabilistic queries on uncertain data. Finally, we present the related work on k-closest pairs queries in Section 2.4.

2.1 Reverse Nearest Neighbor Queries

First, in Section 2.1.1, we present the algorithms that answer RNN queries in Euclidean space. Then, in Section 2.1.2, we provide a brief description of the existing techniques to answer RNN queries in spatial networks

2.1.1 RNN Queries in Euclidean Space

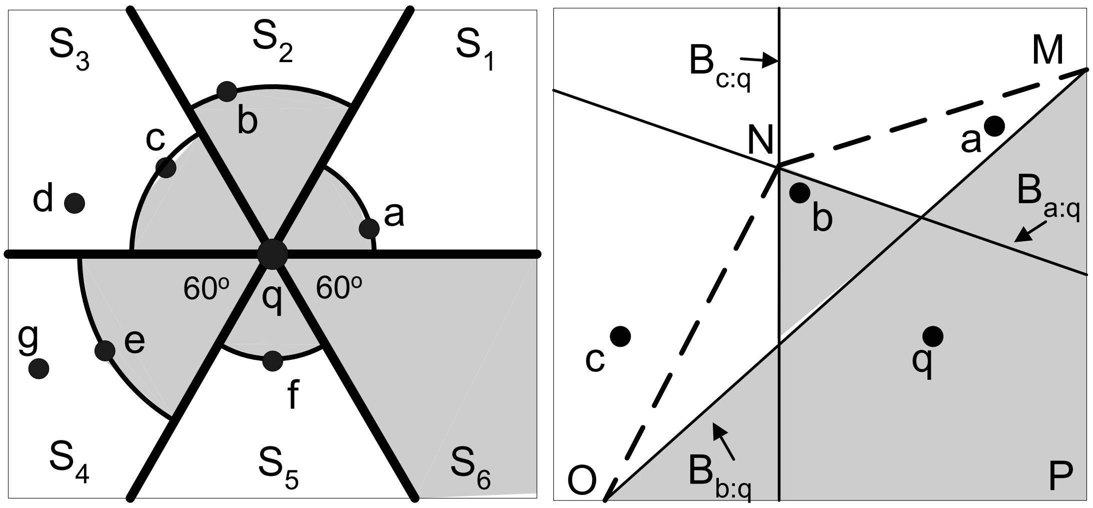
Snapshot RNN Queries: Korn *et al.* [KM00] are the first to study RNN queries. They answer the RNN query by pre-computing a circle for each data object p such that the nearest neighbor of p lies on the perimeter of the circle. RNN of a query q is every point that contains q in its circle. Techniques to improve their work were proposed in [YL01, LNY03].

Now, we briefly describe the existing techniques that do not require precomputation. These techniques have three phases namely *pruning*, *containment* and *verification*. In the pruning phase, the space that cannot contain any RkNN is pruned by applying certain pruning rules. In the containment phase, the objects that lie within the unpruned space are retrieved. These are the possible RkNNs and are called the candidates. In the verification phase, a range query is issued for each candidate object to check if q is one of its k nearest neighbors or not.

First technique that does not need any preprocessing was proposed by Stanoi *et al.* [SAA00]. They solve RNN queries by partitioning the whole space centred at the query q into six equal regions of $60°$ each (S_1 to S_6 in Fig. 2.1(a)). It can be proved that the nearest neighbor to q in each region defines the area that can be pruned. In other words, assume that o is the nearest neighbor of q in a region S_i. Then any user that lies in S_i and lies at a distance greater than $dist(q, o)$ from q cannot be the RNN of q. Fig. 2.1(a) shows nearest neighbors of q in each region and the white area can be pruned. Only the objects that lie in the shaded area can be the RNNs. The RkNN queries can be solved in a similar way, i.e., in each region, the k-th nearest neighbor of q defines the pruned area.

Tao *et al.* [TPL04] proposed TPL that uses the property of perpendicular bisectors to prune the search space. Consider the example of Fig. 2.1(b), where a bisector between q and a is shown as $B_{a:q}$ which divides the space into two half spaces. The half space that contains a is denoted as $H_{a:q}$ and the half space that contains q is denoted as $H_{q:a}$. Any point that lies in the half space $H_{a:q}$ is always closer to a than to q and cannot be the RNN for this reason. Similarly, any point p that lies in k such half spaces cannot be the RkNN. TPL algorithm prunes the space by the bisectors drawn between q and its neighbors in the unpruned area. Fig. 2.1(b) shows the example where the bisectors between q and a, b and c are drawn ($B_{a:q}$, $B_{b:q}$ and $B_{c:q}$, respectively). If $k = 2$, the white area can be pruned because every point in it lies in at least two half spaces.

In the containment phase, TPL retrieves the objects that lie in the unpruned area



(a) Six-regions pruning (b) TPL and FINCH

Figure 2.1: Related techniques

by traversing an R-tree that indexes the locations of the objects. Let m be the number of points for which the bisectors are considered. An area that is the intersection of any combination of k half spaces can be pruned. The total pruned area corresponds to the union of pruned regions by all such possible combinations of k bisectors (a total of $m!/k!(m-k)!$ combinations). Since the number of combinations is too large, TPL uses an alternative approach which has less pruning power but is cheaper. First, TPL sorts the m points by their Hilbert values. Then, only the combination of k consecutive points are considered to prune the space (total m combinations).

Achtert *et al.* [AKK+09] and Emrich *et al.* [EKK+10] propose pruning techniques that can be applied on the rectangles. They use these pruning techniques to prune the intermediate entries of the R-tree that indexes the objects. It was demonstrated that the proposed techniques reduce the number of accessed pages. Moreover, pruning techniques proposed in [EKK+10] are more effective than the pruning techniques of [AKK+09].

Wu *et al.* [WYCT08b] propose an algorithm called FINCH. Instead of using bisectors to prune the objects, they use a convex polygon that approximates the unpruned area. Any object that lies outside the polygon can be pruned. Fig. 2.1(b) shows an example where the shaded area is the unpruned area. FINCH approximates the unpruned area by a polygon $MNOP$. Any point that lies outside this polygon can be pruned. Clearly, the containment checking is easier than TPL because containment can be done in linear time for convex polygons. On the other hand, the area pruned by FINCH is smaller than the area that actually can be pruned.

Continuous RNN Queries: Computation-efficient monitoring of continuous range queries [GL04, LPM02], nearest neighbor queries [MHP05, YPK05, XMA05, ISS03, TPS02] and reverse nearest neighbor queries [BJKS02, XZ06, KMS$^+$07, WYCT08a] has received significant attention. Although there exists work on communication-efficient monitoring of range queries [HXL05] and nearest neighbor queries [HXL05, MPBT05], there is no prior work that reduces the communication cost for continuous RNN queries. Hence, our proposed technique, Lazy Updates, is the only approach that continuously monitors RNN queries and saves the communication cost.

Below, we briefly describe the RNN monitoring algorithms that improve the computation cost.

Benetis *et al.* [BJKS02] present the first continuous RNN monitoring algorithm. However, they assume that velocities of the objects are known. First work that does not assume any knowledge of objects' motion patterns was presented by Xia *et al.* [XZ06]. Their proposed solution is based on the six-regions approach. Consider the examples of Fig. 2.2, where the six-regions approach is applied and the candidate objects are a, b, c, e and f. The shaded area is the pruned area. The results of the RNN query may change in any of the following three scenarios:

1. the query or one of the candidate objects changes its location
2. the nearest neighbor of a candidate object is changed (an object enters or leaves the circles shown in Fig. 2.2)
3. an object moves into the unpruned region (the areas shown in white in Fig. 2.2)

Xia *et al.* [XZ06] use this observation and propose a solution for continuous RNN queries based on the six-regions approach. They answer RNN queries by monitoring six

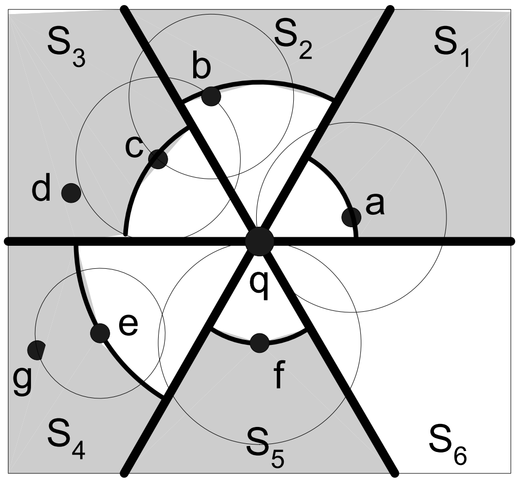


Figure 2.2: Filtering and verification

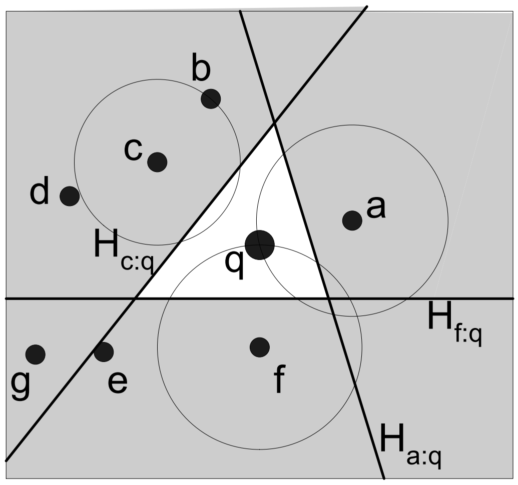


Figure 2.3: Filtering and verification

pie-regions (the white areas in Fig. 2.2) and the circles around the candidate objects that cover their nearest neighbors.

Kang *et al.* [KMS+07] use the concept of half space pruning and apply the same observation that the results may change in any of three scenarios mentioned above (please see the three scenarios shown above and consider Fig. 2.3 instead of Fig. 2.2). They continuously monitor the RNN queries by monitoring the unpruned region (white area in Fig. 2.3) and the circles around the candidate objects that cover their nearest neighbors. The proposed approach uses a grid structure to store the locations of the objects and queries. They mark the cells of the grid that lie or overlap with the area to be monitored. Any object movement in these cells triggers the update of the results.

Wu *et al.* [WYCT08a] are the first to propose a solution for continuous monitoring of RkNN which is similar to the six-regions based RNN monitoring presented in [XZ06]. Wu *et al.* [WYCT08a] issue k nearest neighbor (kNN) queries in each region instead of single nearest neighbor queries. The kNNs in each region are the candidate objects and they are verified if q is one of their k closest objects. To monitor the results, for each candidate object, they continuously monitor the circle around it that contains k nearest neighbors.

Recall that Lazy Updates which we presented in Chapter 3 uses safe regions to reduce the costs. The concept of safe regions has been used in many existing techniques for continuous kNN queries [SR01, PXK$^+$02, ZZP$^+$03, NZTK08, HCLZ09, HXL05] and continuous range queries [ZZP$^+$03, HXL05]. However, these techniques are not applicable for RkNN queries.

2.1.2 RNN Queries in Spatial Networks

Safar *et al.* [SET09] study the snapshot RNN queries in spatial networks. They use Network Voronoi Diagram (NVD) [OBSC99] to efficiently process the RNN queries in spatial networks. A Network Voronoi Diagram (NVD) is similar to a Euclidean space Voronoi Diagram in the sense that every point in each Voronoi cell is closer to the generator point of the cell than any other point. However, a NVD considers minimum network distances instead of Euclidean distances between the points. More specifically, a Voronoi cell in a NVD is the set of nodes and edges that are closer to the generator point (in terms of minimum network distance) than any other point. Safar *et al.* [SET09] use the properties of NVD to efficiently process the RNN queries in network. In a following work [TTS09], they extend their technique to answer RkNN queries and reverse k furthest neighbor queries in spatial network. Please note that their technique cannot be extended to answer continuous RNN queries because the NVD changes as the locations of underlying points change. It is computationally expensive to update NVD whenever the underlying data set changes.

Sun *et al.* [SJLS08] study the continuous monitoring of RNN queries in spatial networks. The main idea is that for each query a multi-way tree is created that helps in defining the monitoring region. Only the updates in the monitoring region affect the results. Their approach is only applicable for the bichromatic RNN queries. Moreover, the proposed approach assumes that the query points do not move. The extension to the case when the query point is also moving is either non-trivial or inefficient because the multi-way trees may be changed as the query points move.

Guohui *et al.* [LLL$^+$10] propose an algorithm to continuously monitor RkNN queries based on dual layer multiway tree (DLM tree). They present several lemmas to obtain a region that is required to be monitored in order to efficiently answer RkNN queries. This monitoring region is represented by a DLM tree to improve the performance.

2.2 Continuous Range Queries

In Section 2.2.1, we present the existing techniques to answer range queries in Euclidean space. In Section 2.2.2, we give an overview of the related techniques for spatial networks.

2.2.1 Range Queries in Euclidean Space

Continuous monitoring of spatial queries has been extensively studied in the recent past [MPBT05, TPS02, HXL05, XMA05, MHP05, ISS03, YPK05, ZZP$^+$03]. Prabhakar *et al.* [PXK$^+$02] proposed velocity constrained indexing and query indexing for continuous evaluation of static queries over moving objects. Mokbel *et al.* [MXA04] introduced an algorithm (SINA) for evaluating a set of concurrent spatial queries, which reduces the overall cost by shared execution and incremental evaluation.

Several distributed processing techniques to continuously monitor range queries have also been proposed [GL04, CHC04, WW06, WZK06]. Gedik *et al.* [GL04] introduce a technique called MobiEyes, which reduces the computation load on the server and communication costs between the clients and the server by delegating some computation load to the client objects (e.g., mobile devices). In [GWYL04], the authors propose a motion adaptive indexing scheme that uses the concept of motion sensitive bounding boxes to model moving objects and queries. Hu *et al.* [HXL05] propose a generic framework to monitor continuous range queries and kNN queries over moving objects. They define the safe zones for each object such that the query results remain unchanged if the object does not leave the region. However, their approach is not designed for moving queries. Wu *et al.* [WCY06] use a new query indexing method called CES-based indexing to minimize the total query evaluation time.

Recall that our algorithm to answer moving range queries uses a safe zone to efficiently process the queries. Several other techniques also propose to construct safe zones for moving kNN queries [ZL01, SR01, ZZP$^+$03, NZTK08, HCLZ09] and moving window queries [ZZP$^+$03]. However, to the best of our knowledge, there does not exist any safe zone based technique to continuously monitor moving circular range queries. We next show that the existing work cannot be extended to monitor moving circular range queries continuously.

Tao *et al.* [TP02] introduce Time-Parameterized queries (TP queries). A TP query assumes that the motion pattern (e.g., path and speed) of the query is known and retrieves the current results along with a future time at which the current results will become invalid. A TP query also reports the object that invalidates the results. In [TP02], the techniques to answer TP kNN queries, TP window queries and TP join queries are presented.

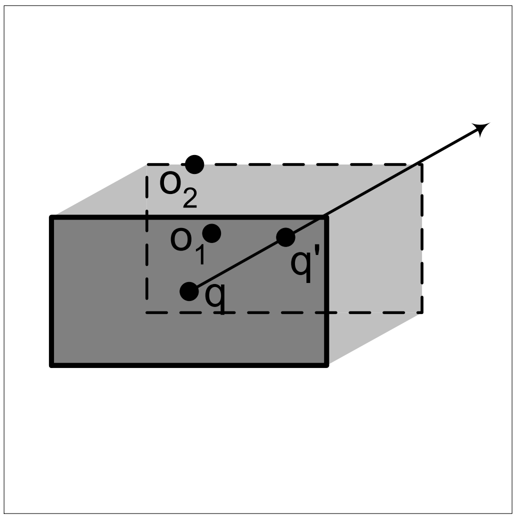


Figure 2.4: A time-parameterized window query

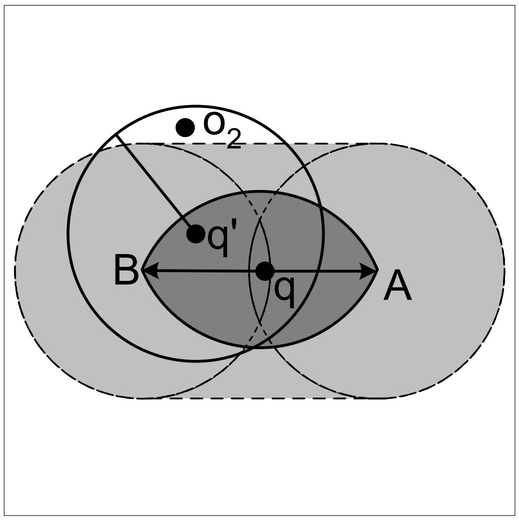


Figure 2.5: TP circular queries cannot be used to construct safe zone

Fig. 2.4 shows an example of a window query where the current location of the query is q and its window is shown with a solid line (the search space is shown in a dark shade). The current result of the window query q is the object o_1. A TP window query is issued to find the object that invalidates the current result when the query is moving in the direction shown by the arrow. The query returns the object o_2 as it invalidates the current result when the query reaches the location q'. In other words, when the query reaches q', it has objects $o1$ and o_2 within its window and not only $o1$. The minimal area searched by the TP query is shown shaded in Fig. 2.4.

Based on TP queries, Zhang *et al.* [ZZP+03] present a solution to continuously monitor kNN queries and the window queries. They use TP queries to identify the safe zones for moving queries. The algorithm starts by assuming that the whole space is the safe zone. TP queries are then issued towards the corners of the current safe zone. If a TP query retrieves an object that has not already been considered, the safe zone is trimmed using that object (for details, see [ZZP+03]); otherwise, the corner is marked as confirmed. The algorithm terminates when all the corners are confirmed.

We note that there does not exist any reported work on TP circular range queries and the technique presented in [ZZP+03] cannot be applied to such queries. Even if the technique to answer TP window queries are extended to answer the TP circular range queries, the TP circular range queries cannot be used to construct the safe zone. The reason is as follows. The key observation used in the technique presented in [ZZP+03] is that if none of the TP queries issued towards corners of a region returns a new object, the region is guaranteed to be the safe zone. This observation does not hold for the moving circular range queries. Consider the example in Fig. 2.5 where the current region is shown dark shaded. The TP range queries are issued towards each of the two corners A and B and they search the space shown shaded in the figure. No object is returned by either of the TP range queries. However, the region cannot be guaranteed to be the safe zone. Consider that the query moves to the location q'. Then the object o_2 lies within its range, which invalidates the results.

2.2.2 Range Queries in Spatial Networks

Significant research attention has been given to developing techniques for spatial queries in road networks. kNN queries [JKPT03, CC05, KS04b, KS04a, PZMT03, MYPM06, SKS02, SY03] and range queries [SPP+08, PZMT03, LDH06, WZ08] are among the most studied spatial queries in road networks. Chen *et al.* [CSZY09] study the path k-NN queries that returns kNNs with respect to the shortest path connecting the destination and the user's current location.

Papadias *et al.* [PZMT03] propose a framework to support nearest neighbor queries, closest pairs queries, range queries and distance joins on a road network. However, they assume that the queries and the objects have fixed positions in the spatial network. Wang *et al.* [WZ08] propose a solution to answer static range queries over moving objects. They utilize a disk resident R-tree to store the network and a grid structure to store the positions of moving objects. The main idea is to first find the edges that may contain the objects within the range and then the grid cells that overlap with the edges are used to retrieve the objects. Liu *et al.* [LDH06] present a distributed processing technique to solve the moving range queries over moving objects. Their approach relies on the computation power of the moving objects and each moving object reports to the server when it affects the results of one or more queries. Stojanovic *et al.* [SPP$^+$08] propose technique for continuous monitoring of range queries over moving objects. The range of the query may be defined by a user selected area, a map window, a polygon, a circle or a part of the road segment.

Kriegel *et al.* [KKR08] study the problem of proximity monitoring in road networks. Given a proximity threshold ϵ and a set of moving objects, a server responsible for proximity monitoring continuously reports the pairs of objects that are within a distance ϵ to each other. Küpper *et al.* [KT06] propose a technique for the same problem in Euclidean space. Both of the techniques assign each moving object a region such that as long as the object remains within this region it does not need to report its location to the server. Note that these techniques can be adopted to answer the distance based range queries by setting the proximity threshold to r and considering only the pairs of objects that contain the query object q. However, the focus of these techniques is to reduce the communication cost between the moving objects and the server. On the other hand, the focus of our technique is to minimize the computation time. Moreover, our framework is suitable for both the client-server model and the local computation model.

2.3 Probabilistic Spatial Queries on Uncertain Data

Recently, a lot of work has been dedicated to uncertain databases (see The TRIO system [Wid05], The ORION project [CPK03] and the references therein). Query processing on uncertain databases has gained significant attention in last few years especially in spatio-temporal databases.

In [CXP+04], the authors develop index structures to querying uncertain interval effectively. They are the first to study probabilistic range queries. In [TCX+05], the authors propose access methods designed to optimize both the I/O and CPU cost of range queries on multi-dimensional data with arbitrary probability density functions. The concept of probabilistic similarity joins on uncertain objects is first introduced in [KKPR06] which assigns a probability value to each object pair indicating the likelihood that it belongs to the result set. Ranking and thresholding probabilistic spatial queries are studied in [DYM+05]. A thresholding probabilistic query is to retrieve the objects qualifying the spatial predicates with probability greater than a given threshold. Similarly, a ranking probabilistic query retrieves the objects with the highest probabilities to qualify the spatial predicates.

Nearest neighbor queries on uncertain objects have also received significant research attention [BSI08, CCMC08, KKR07]. Beskales *et al.* [BSI08] propose a technique to return objects with the highest marginal probability of being the nearest neighbor of a query. They also propose extensions to handle dependencies among the objects and to answer threshold queries. Cheng *et al.* [CCX09] study the probabilistic kNN query which computes the probabilities of sets of k objects for being the closest to a given query point. They also propose techniques to answer probabilistic threshold kNN query that returns the sets of k objects that satisfy the query with probabilities higher than a given threshold. Potamias *et al.* [PBGK10] study k nearest neighbor queries on uncertain graphs.

A probabilistic skyline model is proposed in [PJLY07] along with two effective algorithms to answer probabilistic skyline queries. Zhang *et al.* [ZZL+11] study the problem of top-k skyline on uncertain data. Atallah *et al.* [AQ09, AQY11] propose novel sub-quadratic algorithm to compute the skyline probabilities of all the objects. Lin *et al.* [LZZC11] propose to use *lower orthant order* in modeling stochastic dominance relationship among the objects. Their proposed model guarantees to provide a minimum set of candidates for the family of multiplicative decreasing scoring functions.

It is important to mention that two other approaches to answer probabilistic reverse nearest neighbor queries have been proposed [L09, BEK+11]. Lian *et al.* [L09] independently proposed a technique to answer RNN queries on uncertain data at around the same time when we proposed our probabilistic RNN queries algorithm [CLW+10]. In their approach, they approximate the uncertain objects by circular regions whereas we approximate the uncertain objects by rectangular regions. Based on these circular regions, they propose some pruning techniques to shortlist a set of candidate objects. Recently, Bernecker *et al.* [BEK+11] propose a solution to answer RkNN queries on uncertain data. Like our technique, they also approximate the uncertain objects by rectangular regions. They propose new pruning rules and also present the techniques to answer RkNN queries for $k > 1$. Their experimental results demonstrate that their proposed approach is the most efficient approach till date. The experimental study also demonstrates that our approach performs better than the approach presented in [L09].

2.4 k Closest Pairs Queries

The k closest pairs query is a special case of the score-based top-k pairs queries (see Chapter 7). The problem of k closest pairs queries has received significant research attention by the computational geometry community (see [Smi97] for a nice survey). Below, we give an overview of the previous work in the context of spatial databases.

Hjaltason *et al.* [HS98] are the first to study the problem of closest pairs in the context of spatial databases. They propose incremental distance joins where two data sets are joined and the pairs are output incrementally according to the distances between them. Each data set is indexed by an R-tree and a priority queue is used to store the intermediate entry pairs. While the proposed solution has a nice feature that it returns the pairs incrementally, its priority queue size may be prohibitively large. For this reason, a part of the priority queue is kept in main memory and remaining elements are stored in secondary memory as a number of linked lists.

Corral *et al.* [CMTV00] propose several algorithms for k closest pairs queries. Similar to the previous algorithm [HS98], they also index the data sets by R-trees. They use distance bounds to prune the intermediate node pairs. They observe that the performance of their algorithm largely depends on the overlap factor of the two data sets. They also extend their algorithms to answer non-chromatic k closest pairs queries and k furthest pairs queries. It is important to note that although the amount of the memory used by their algorithm is small as compared to the memory usage of the algorithm proposed in [HS98], there is no guarantee on the amount of the main memory usage (e.g., the size of the heap can be $O(V)$ where V is the total number of possible pairs).

Yang *et al.* [YL02] proposed a data structure to further improve the k closest pairs algorithm. Their algorithm works for the case when all the pairs have unique distances [SZS03]. Several variants of k closest pairs queries have also been studied in [UMY07, AP05, SZS03, QTJ$^+$08].

Their proposed data structure is similar to R-tree except that it stores for each object in one data set the distance to its nearest neighbor dnn from the other data set. The intermediate nodes store the minimum and maximum dnn of its child entries. Their algorithm is based on the theorem that k closest pairs can be obtained by joining k objects from each data set with minimum dnn.

Note that all of the above mentioned algorithms are mainly designed for the bichromatic k closest pairs queries. Some variants of k closest pairs queries have also been studied. More specifically, continuous monitoring of exclusive k closest pairs [UMY07], approximate k closest pairs in high dimensional data [AP05] and constrained k closest pairs [SZS03, QTJ+08] have been studied.

Chapter 3

Lazy Updates: Continuously Monitoring RkNN Queries

In this chapter, we present our techniques to continuously monitor reverse k nearest neighbors queries in Euclidean space and in spatial networks. Our technique is called Lazy Updates and it significantly reduces CPU cost as well as communication cost of the system. The research presented in this chapter was published in [CLZ+09, CZL+11].

3.1 Overview

As defined in Chapter 1, a reverse k nearest neighbors (RkNN) query q retrieves all the data points that have q as one of their k nearest neighbors (k closest points). We use RNN queries to refer to RkNN queries for which $k = 1$. Consider the example of Fig. 3.1 where q is a RNN query in Euclidean space. The nearest neighbor (the closest object in Euclidean space) of q is o_1. However, o_1 is not the RNN of q because the closest point of o_1 is not q. The RNNs of q are o_3 and o_4 because q is the nearest neighbor for both of these points.

RNN has received considerable attention [KM00, SAA00, BJKS02, SFT03, LNY03, TPL04, TYM06, YPMT05, YM07, WYCT08b] from database research community based

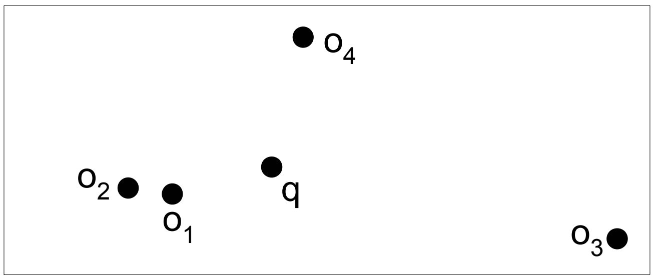


Figure 3.1: o_3 and o_4 are RNNs of q in Euclidean space

on the applications such as decision support, location based service, resource allocation, profile-based management, etc.

With the availability of inexpensive mobile devices, position locators and cheap wireless networks, location based services are gaining increasing popularity. Consider the example of a battlefield. A backup army unit may issue a RNN query to monitor other units for which this is the closest army unit. Such army units may seek help from the backup army unit in the case of an emergency event. Therefore the backup army unit may issue a RNN query to retrieve such army units and may observe their status from time to time (e.g., current location, ammunition etc.).

Note that in the above example, the query objects and the data objects both belong to the same type of objects (i.e., army units). Such queries are called *monochromatic* queries. The queries where the query objects and the data objects belong to two different types of objects are called *bichromatic* queries (formally defined in Section 3.2.1). Consider the example of a user that needs a taxi and sends her location to a taxi company's dispatch center. The company owns several taxis and wants to send this job to a taxi for which she is the closest passenger. Hence, the company notifies the taxis that are among the bichromatic RNNs of the user. Cabspotting[1] and Zhiing[2] are two examples of such location based services.

Other examples of location based services include location based games, traffic monitoring, location based SMS advertising, enhanced 911 services and army strategic plan-

[1]http://cabspotting.org/faq.html
[2]http://www.zhiing.com/how.php

ning etc. These applications may require continuous monitoring of reverse nearest moving objects. For instance, in reality games (e.g., BotFighters, Swordfish), players with mobile devices search for other mobile devices in neighborhood. For example, in the award winning game BotFighters, a player gets points by shooting other nearby players via mobiles. In such an application, some players may want to continuously monitor her reverse nearest neighbors in order to avoid being shot by other players.

Driven by such applications, the continuous monitoring of reverse nearest neighbors has been investigated and several techniques have been proposed recently [BJKS02, KMS+07, WYCT08a, XZ06, SJLS08] in the light of location-based services. The existing continuous monitoring techniques [BJKS02, KMS+07, WYCT08a, XZ06] adopt two frameworks based on different applications. In [BJKS02], the velocity of each object is assumed to be explicitly expressed while [KMS+07, WYCT08a, XZ06] deal with a general situation where the object velocities may be impossible to be explicitly expressed. Our research is based on the general situation; that is, object velocities are not explicitly expressible.

The techniques in [KMS+07, WYCT08a, XZ06] adopt a two-phase computation. In the *filtering* phase, objects are pruned by using the existing pruning paradigms from [SAA00, TPL04] and the remaining objects are considered as the candidate objects. In the *verification* phase, every candidate object for which the query is its closest point is reported as the RNN. To update the results, at each time-stamp, if the set of candidate objects is detected to be unchanged then only the verification phase is called to verify the results. Nevertheless, both the filtering and verification phases are required if one of the candidate objects changes its location or other objects move into the candidate region. Similarly, a set of candidate objects is needed to be re-computed (recall filtering) if the query changes its location.

As mentioned earlier, previous techniques [KMS+07, XZ06, WYCT08a] require expensive filtering if the query or any of the candidate objects changes its location. Our initial experiment results show that the cost of verification phase is much lower than the cost of filtering phase. In our technique, we assign each query and object a safe region. The safe region is a rectangular area for the queries in Euclidean space and is a subset of spatial network for the queries in spatial networks. The filtering phase for a query is not required as long as the query and its candidate objects remain in their corresponding safe regions. This significantly reduces the computation time of continuously monitoring RkNN queries.

As a by-product, our proposed framework also significantly reduces the communication cost in a client-server architecture. In the existing techniques, every object reports its location to the server at every time-stamp regardless whether query results will be affected or not. Consequently, such a computation model requires transmission of a large number of location updates; doing this has a direct impact on the wireless communication cost and power consumption - the most precious resources in mobile environment [HXL05]. In our framework, each moving object reports its location update only when it leaves the region. This significantly saves the communication costs.

Depending on the users' needs, applications may require RNN queries to be monitored in Euclidean space or in spatial networks (e.g., a road network). While several algorithms have been proposed to monitor RNN queries in Euclidean space there does not exist any algorithm that efficiently updates RNNs in spatial networks after the objects and queries change their locations. In this chapter, we present efficient algorithms to monitor RNN queries in Euclidean space as well as in spatial networks.

Below, we summarize our contributions:

Query processing in Euclidean space

1. We present a framework for continuously monitoring RNN together with a novel set of effective pruning and efficient increment computation techniques. It not only reduces the total computation cost of the system but also reduces the communication cost.

2. We extend our algorithm for the continuous monitoring of RkNN. Our algorithm can be used to monitor both *mono-chromatic* and *bichromatic* RkNN (to be for-

mally defined in Section 3.2.1).

3. We provide a rigid analysis on the computation and communication costs of our algorithm that helps us to understand the effect of the size of the safe region on the costs of our algorithm.

4. Our extensive experiments demonstrate that the developed techniques outperform the previous algorithms by an order of magnitude in terms of computation cost and communication cost.

Query processing in spatial networks

5. We are first to present a continuous RNN monitoring algorithm for moving objects and queries in spatial networks. The proposed algorithm is computationally efficient and has low communication cost.

6. We show that our technique can be easily extended to monitor *mono-chromatic* and *bichromatic* RNN queries. The algorithm can also be extended to continuously monitor RkNN queries.

7. We conduct extensive experiments on a real road network and demonstrate that our algorithm gives an order of magnitude improvement over an algorithm that does not use the safe regions.

The rest of this chapter is organized as follows. In Section 3.2, we give the problem statement and describe the motivation. Section 3.3 presents RNN monitoring techniques for Euclidean space including a detailed theoretical analysis. Section 3.4 presents our technique for continuously monitoring RNN queries and its variants in spatial networks. The experiment results are reported in Section 3.5. Section 3.6 summarizes the chapter.

3.2 Background Information

In this section, we first formally define the problem in Section 3.2.1. Then, we give a brief description of related work in Section 3.2.2 to better explain the motivation of our work. The motivation of our approach is presented in Section 3.2.3.

3.2.1 Problem Definition

There are two types of RkNN queries [KM00] namely, *mono-chromatic* and *bichromatic* RkNN queries. Below we define both.

Monochromatic RkNN query: Given a set of points P and a point $q \in P$, a monochromatic RkNN query retrieves every point $p \in P$ s.t. $dist(p, q) \leq dist(p, p_k)$ where $dist()$ is a distance function, and p_k is the kth nearest point to p according to the distance function $dist()$. In Euclidean space, $dist(x, y)$ returns the Euclidean distance between any two points x and y. In spatial networks, $dist(x, y)$ returns the minimum network distance between any two points lying on the spatial network.

Note that, in such queries, both the data objects and the query objects belong to the same class of objects. Consider an example of the reality game BotFighters, where a player issues a query to find other players for whom she is the closest person.

Bichromatic RkNN query:

Given two sets O and P each containing different types of objects, a bichromatic RkNN query for a point $q \in O$ is to retrieve every object $p \in P$ such that $dist(p, q) \leq dist(p, o_k)$ where o_k is the kth nearest point of p in O according to the distance function $dist()$.

In contrast to monochromatic queries, the query and data objects belong to two different classes. Consider the example of a battlefield where a medical unit might issue a bichromatic RNN query to find the wounded soldiers for whom it is the closest medical unit.

Our main focus in this chapter is to present the techniques to continuously monitor monochromatic queries. However, we show that the proposed techniques can be easily extended to answer bichromatic queries. In rest of this chapter, we use RNN query to refer to a monochromatic RNN query unless mentioned otherwise.

3.2.2 Most closely related techniques

To better illustrate the motivation of our work, we briefly describe most relevant techniques to monitor RNN queries in Euclidean space. First, we present pruning techniques for *snapshot* RNN queries.

Snapshot RNN Queries

Stanoi *et al.* [SAA00] solve RNN queries by partitioning the whole space centred at the query q into six equal regions of $60°$ each (S_1 to S_6 in Fig. 3.2). It can be proved that only the nearest point to q in each partition can possibly be the RNN. This also means that, in two-dimensional space, there are at most six possible RNNs of a query. Consider the region S_3 where c is the nearest object to q and d cannot be the RNN because its distance to c is smaller than its distance to q. This can be proved by the triangle Δqcd where $\angle dqc \leq 60°$ and $\angle dcq \geq 60°$, hence $dist(d, c) \leq dist(d, q)$. Fig. 3.3 shows the area (shown shaded) that cannot contain RNN of q.

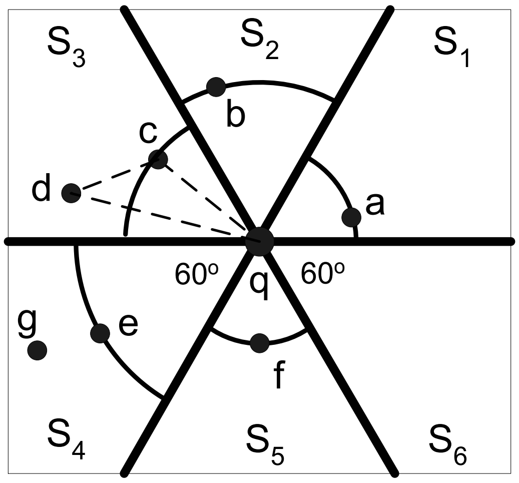


Figure 3.2: Pruning based on six-regions

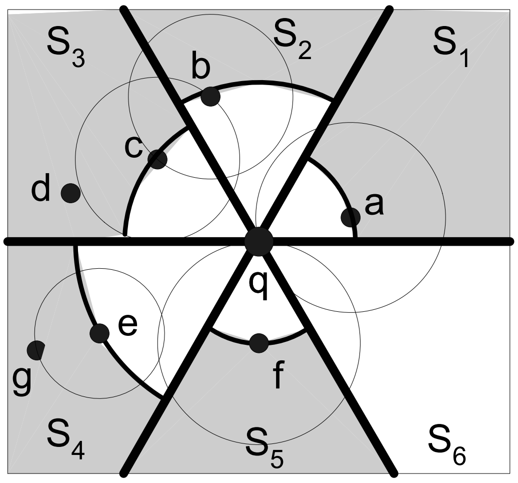


Figure 3.3: Filtering and verification

In *filtering* phase, the candidate RNN objects (a, b, c, e and f in our example) are selected by issuing nearest neighbor queries in each region. In *verification* phase, any candidate object for which q is its nearest neighbor is reported as RNN (a and f). We call this approach *six-regions pruning* approach.

Tao *et al.* [TPL04] use the property of perpendicular bisectors to answer RkNN queries. Consider the example of Fig. 3.4, where a bisector between q and c is shown that divides the space into two half spaces (the shaded half space and the white half space). Any point that lies in the shaded half space $H_{c:q}$ is always closer to c than to q and cannot be the RNN for this reason. Their algorithm prunes the space by the half spaces drawn between q and its neighbors in the unpruned region. Fig. 3.5 shows the example where half spaces between q and a, c and f ($H_{a:q}$, $H_{c:q}$ and $H_{f:q}$, respectively) are shown and the shaded area is pruned. Then, the candidate objects (a, c and f) are verified as RNNs if q is their closest object. We call this approach *half space pruning* approach. It is shown in [TPL04] that the half space pruning is more powerful than the six-regions pruning and it prunes larger area (compare the shaded areas of Fig. 3.3 and Fig. 3.5).

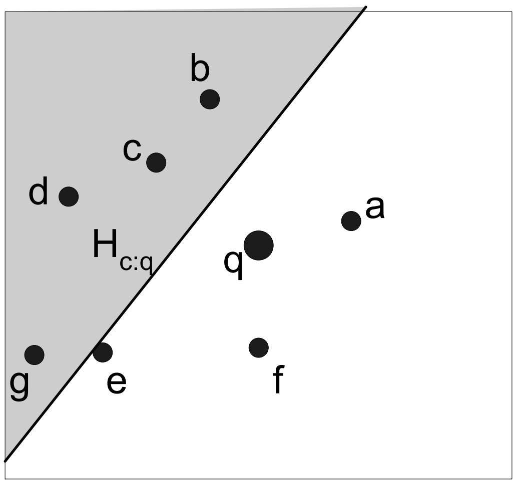


Figure 3.4: Pruning based on half spaces

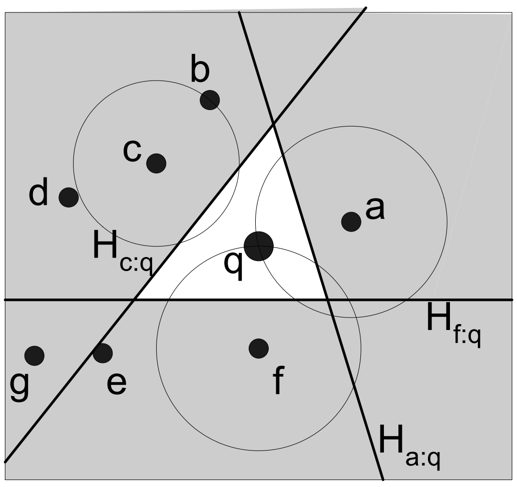


Figure 3.5: Filtering and verification

Continuous RNN Queries

Xia *et al.* [XZ06] propose a solution based on the six-regions approach. Consider the examples of Fig. 3.3, the results of the RNN query may change in any of the following three scenarios:

1. the query or one of the candidate objects changes its location

2. the nearest neighbor of a candidate object is changed (an object enters or leaves the circles shown in Fig. 3.3)

3. an object moves into the unpruned region (the areas shown in white in Fig. 3.3)

Xia *et al.* [XZ06] use this observation and propose a solution for continuous RNN queries based on the six-regions approach. They answer RNN queries by monitoring six pie-regions (the white areas in Fig. 3.3) and the circles around the candidate objects that cover their nearest neighbors.

Kang *et al.* [KMS+07] use the concept of half space pruning and apply the same observation that the results may change in any of three scenarios mentioned above (please see the three scenarios shown above and consider Fig. 3.5 instead of Fig. 3.3). They continuously monitor the RNN queries by monitoring the unpruned region (white area in Fig. 3.5) and the circles around the candidate objects that cover their nearest neighbors. The proposed approach uses a grid structure to store the locations of the objects and queries. They mark the cells of the grid that lie or overlap with the area to be monitored. Any object movement in these cells triggers the update of the results.

Wu *et al.* [WYCT08a] are the first to propose a solution for continuous monitoring of RkNN which is similar to the six-regions based RNN monitoring presented in [XZ06]. Wu *et al.* [WYCT08a] issue k nearest neighbor (kNN) queries in each region instead of single nearest neighbor queries. The kNNs in each region are the candidate objects and they are verified if q is one of their k closest objects. To monitor the results, for each candidate object, they continuously monitor the circle around it that contains k nearest neighbors.

3.2.3 Motivation

First, we briefly describe limitations of existing techniques that monitor RNNs in Euclidean space. Both the six-regions [XZ06] and the half space [KMS+07] based solutions have two major limitations.

1. As illustrated in the three scenarios presented in Section 3.2.2, the existing techniques are sensitive to object movement. If a query or any of its candidate objects changes its location, filtering phase is called again which is computationally expensive. For example, if a query is continuously moving, at each timestamp both of the approaches will have to compute the results from scratch. For example, in the half space based approach, the half spaces between q and its previous candidates are redrawn and the pruning area is adjusted. In our initial experiments, we find that the cost of redrawing the half spaces (and marking and unmarking the relevant cells) is computationally almost as expensive as the initial computation of the results.

2. The previous techniques require every object to report its exact location to the server at every timestamp regardless of whether it affects the query result or not. This has a direct impact on the two most precious resources in mobile environment, wireless communication cost and power consumption. Ideally, only the objects that affect the query results should report their locations. For example, in Fig. 3.5, as long as objects d, e and g do not enter into the white region or the three circles, they do not affect the results of the query.

Motivated by these, we present a framework that provides a computation and communication efficient solution. Note that, in some applications, the clients may have to periodically report their locations to the server for other types of queries. In this case, saving the communication cost is not possible. Nevertheless, our framework significantly reduces the computation costs for such applications[3].

3.3 Query Processing in Euclidean Space

In this section, we present our technique to continuously monitor RNN queries in Euclidean space. In Section 3.3.1, we present the framework of our proposed technique. A

[3] In this chapter, we present our technique assuming that the clients send their locations only for the RkNN query. For the case when the clients periodically send their locations for other types of queries, our techniques can be easily applied. The only change is that the safe regions are stored on the server which ignores the location updates from the objects that are still in their safe regions. Experiment results shown in Section 3.5 show the superiority of our approach for both of the cases.

set of novel pruning techniques is presented in Section 3.3.2. Our continuous RNN monitoring algorithm is presented in Section 3.3.3. In Section 3.3.4, we present a detailed theoretical analysis to analyse the computation and communication cost of our proposed algorithms. We present the extensions of our approach to monitor other variants of RNN queries in Section 3.3.5.

3.3.1 Framework

Each moving object and query is assigned a safe region of a rectangular shape. Although other simple shapes (e.g., circles) could be used as safe regions, we choose the safe region of a rectangular shape mainly because defining effective pruning rules is easier for the rectangular safe regions. The clients may use their motion patterns to assign themselves better safe regions. However, we assume that such information is not utilized by the clients or the server because we do not assume any knowledge about the motion pattern of the objects.

In our framework, the server recommends the side lengths of the safe regions (a system parameter) to the clients. When a client leaves its safe region, the client assigns itself a new safe region such that it lies at the center of the safe region and reports this safe region to the server.

An object reports its location to the server only when it moves out of its safe region. Such updates issued by the clients (objects) are called *source-initiated* updates [HXL05]. In order to update the results, the server might need to know the exact location of an object that is still in its safe region. The server sends a request to such object and updates the results after receiving its exact location. Such updates are called *server-initiated* updates [HXL05].

If an object stops moving (e.g., a car is parked), it notifies the server and the server reduces its safe region to a point until it starts moving again. Client devices such as GPS can be programmed to notify the server when the device stops moving (e.g., the GPS notifies the server if the car engine is turned off or if the car did not move in last T time units).

In the previous approaches [XZ06, KMS+07], the pruned area becomes invalid if the query point changes its location. On the other hand, in our framework, the query is also assigned with a safe region and the pruned area remains valid as long as the query and its candidate objects remain in their respective safe regions and no other object enters in the unpruned region. Although the query is also assigned with a safe region, it reports its location at every timestamp. This is because its location is important to compute the exact results and a server-initiated update would be required (in most of the cases) if it does not report its location itself. Moreover, the number of queries in the system is usually much smaller than the number of objects. Hence, the location updates by the queries do not have significant effect on the total communication cost.

Table 3.1 defines the notations used throughout this section.

Notation	Definition
$B_{x:q}$	a perpendicular bisector between point x and q
$H_{x:q}$	a half space defined by $B_{x:q}$ containing point x
$H_{q:x}$	a half space defined by $B_{x:q}$ containing point q
$H_{a:b} \cap H_{c:d}$	intersection of the two half spaces
$A[i]$	value of a point A in the i^{th} dimension
$maxdist(x, y)$	maximum distance between x and y (each of x and y is either a point or a rectangle)
$mindist(x, y)$	minimum distance between x and y (each of x and y is either a point or a rectangle)
R_{fil}, R_{cnd}, R_q	rectangular region of the filtering object, candidate object and query, respectively
$R_H[i]$	highest coordinate value of a rectangle R in i^{th} dimension
$R_L[i]$	lowest coordinate value of a rectangle R in i^{th} dimension

Table 3.1: Notations

Like existing work on continuous spatial queries [MHP05, KMS+07, XZ06], we assume that the errors due to the measuring equipments are insignificant and can be ignored. Our continuous monitoring algorithm consists of the following two phases.

Initial computation: When a new query is issued, the server first computes the set of candidate objects by applying pruning rules presented in Section 3.3.2. This phase is called *filtering* phase. Then, for each candidate object, the server verifies it as RkNN if the query is one of its k closest points. This phase is called *verification* phase.

Continuous monitoring: The server maintains the set of candidate objects throughout the life of a query. Upon receiving location updates, the server updates the candidate set if it is affected by some location updates. Otherwise, the server calls verification module to verify the candidate objects and reports the results.

3.3.2 Pruning Rules

In this section, we present novel pruning rules for RNN queries that can be applied when locations of the objects are unknown within their rectangular regions. Although the proposed pruning rules work in any multidimensional space, to keep the discussion simple, we focus on two dimensional space in this section. The pruning rules for higher dimensionality are similar and are presented in Chapter 5.

We also remark that the proposed pruning rules can be applied on the minimum bounding rectangles of the spatial objects that have irregular shapes (in contrast to the assumption that the spatial objects are points). In Section 3.3.5, we extend the pruning rules for RkNN queries.

Throughout this section, an object that is used for pruning other objects is called a *filtering* object and the object that is being considered for pruning is called a *candidate* object.

Half space Pruning

First, we present the challenges in defining this pruning rule by giving an example of a simpler case where the exact location of a filtering object p is known but the exact location of q is not known on a line MN (shown in Fig. 3.6). Any object x cannot be the RNN of q if $mindist(x, MN) \geq dist(x, p)$ where $mindist(x, MN)$ is the minimum distance of x from the line MN. Hence, the boundary that defines the pruned area consists of every point x that satisfies $mindist(x, MN) = dist(x, p)$. Note that for any point x in the space on the right side of the line L_N, $mindist(x, MN) = dist(x, N)$. Hence, in the space on the right side of the line L_N, the bisector between p and the point N satisfies the equation of the boundary (because for any point x on this bisector $dist(x, N) = dist(x, p)$).

Similarly, on the left side of L_M, the bisector between p and M satisfies the condition. In the area between L_M and L_N, a parabola (shown in Fig. 3.6) satisfies the equation of the boundary. Hence the shaded area defined by the two half spaces and the parabola can be pruned. Note that the intersection of half spaces $H_{p:N}$ and $H_{p:M}$ does not define the area correctly. As shown in Fig. 3.6 , a point p' lying in this area may be closer to q than to the point p.

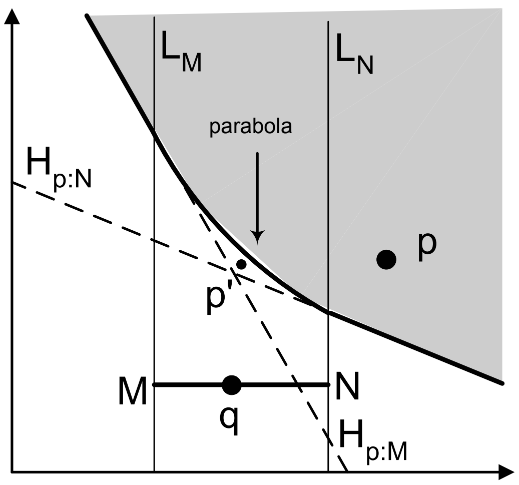


Figure 3.6: Exact location of q on line MN is not known

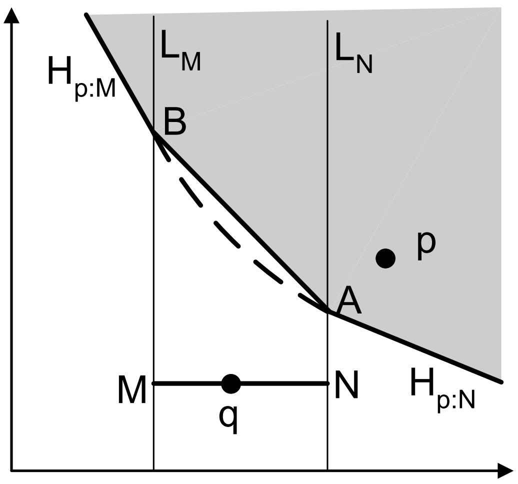


Figure 3.7: Approximation of parabola by a line

Unfortunately, the pruning of the shaded area may be expensive due to presence of the parabola. One solution is to approximate the parabola by a line AB where A is the intersection of $H_{p:N}$ and L_N and B is the intersection of $H_{p:M}$ and L_M. Fig. 3.7 shows the line AB and the pruned area (the shaded area).

Another solution is to move the half spaces $H_{p:M}$ and $H_{p:N}$ such that both pass through a point c that satisfies $mindist(c, MN) \geq dist(c, p)$ (e.g., any point lying in the shaded area of Fig. 3.6). This approximation of the pruning area is tighter if the point c lies on the boundary. Fig. 3.8 shows the half spaces $H_{p:M}$ and $H_{p:N}$ moved to such point c. A half space that is moved is called *normalized* half space and a half space $H_{p:M}$ that is moved is denoted as $H'_{p:M}$. Fig. 3.8 shows the normalized half spaces $H'_{p:M}$ and $H'_{p:N}$ and their intersection can be pruned (the shaded area).

Among the two possible solutions discussed above, we choose normalized half spaces in developing our pruning rules for the following reason. In our relatively simple example, the number of half spaces required to prune the area by using the normalized half spaces is two (in contrast to three lines for the other solution). The difference between this number becomes significant when both the query and the filtering object are represented by rectangles especially in multidimensional space. This makes the pruning by normalized half spaces a less expensive choice.

Now, we present our pruning rule that defines the pruned area by using at most four half spaces in two dimensional space. This pruning rule uses the normalized half spaces between 4 selected pairs of corners of the two rectangles to prune the space. We first give a formal description of our pruning rule and then we briefly describe the reason of its correctness. First, we define the following concepts:

Antipodal Corners. Let C be a corner of rectangle $R1$ and C' be a corner in $R2$. The two corners are called *antipodal corners* if both of the followings hold: i) if C is a corner on the lower side of $R1$ then C' is a corner on the upper side of $R2$ and vice versa; ii) if C is a corner on the right side of $R1$ then C' is a corner on the left side of $R2$ and vice versa.

For example, a lower-left corner of $R1$ is the antipodal corner of the upper-right corner of $R2$. Similarly, an upper-left corner of $R1$ is the antipodal corner of the lower-right corner of $R2$. Fig. 3.9 shows two rectangles $R1$ and $R2$. The corners B and M are two antipodal corners. Similarly, other pairs of antipodal corners are (D, O), (C, N) and (A, P).

Antipodal half space. A half space that is defined by the bisector between two antipodal corners is called *antipodal half space*. Fig. 3.9 shows two antipodal half spaces $H_{M:B}$ and $H_{O:D}$.

Higher and lower midpoints. Let $R1$ and $R2$ be two rectangles. Let $R1_L[i]$ denote

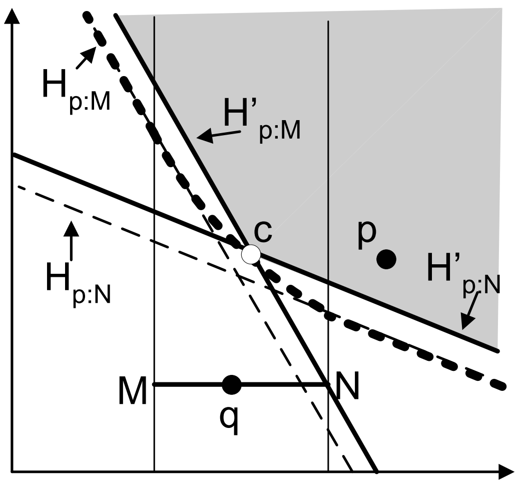


Figure 3.8: Defining pruned region by moving half spaces

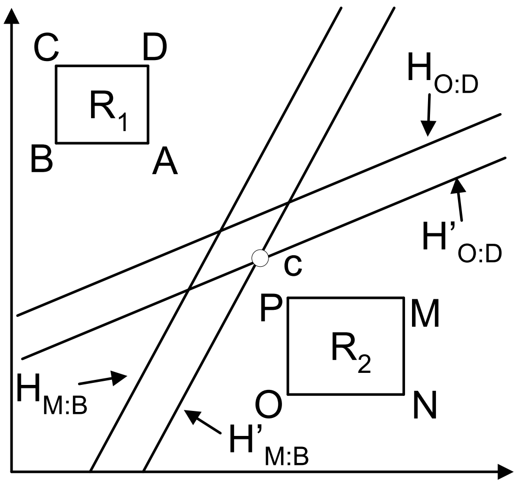


Figure 3.9: Antipodal corners and normalized half spaces

the lowest coordinate value and $R1_H[i]$ denote the highest coordinate value of $R1$ in i^{th} dimension. The higher midpoint $M_H[i]$ of two rectangles $R1$ and $R2$ in i^{th} dimension is $(R1_H[i] + R2_H[i])/2$. Similarly, the lower midpoint $M_L[i]$ of two rectangles $R1$ and $R2$ in i^{th} dimension is $(R1_L[i] + R2_L[i])/2$.

Assume that for a point P, we denote its x and y coordinate values as $P.x$ and $P.y$, respectively. In the example of Fig. 3.9, the higher midpoint of $R1$ and $R2$ along x-axis is $(N.x + A.x)/2$ (see $c.x$). Similarly, the lower midpoint along y-axis is $(O.y + A.y)/2$ (see $c.y$).

Normalized half space. Let B and M be two points in the rectangles $R1$ and $R2$, respectively. The normalized half space $H'_{M:B}$ is a half space defined by the bisector between M and B that passes through a point c such that $c[i] = M_L[i]$ (lower midpoint) for every dimension i for which $B[i] > M[i]$ and $c[j] = M_H[j]$ (higher midpoint) for every dimension j for which $B[j] \leq M[j]$. A normalized antipodal half space can be represented by a mathematical inequality. The interested readers are referred to Chapter 5.

Fig. 3.9 shows the normalized (antipodal) half spaces $H'_{M:B}$ which is obtained by moving the half space $H_{M:B}$ to the point c where $c.x$ is the higher midpoint of the two rectangles along x-axis (because $B.x < M.x$) and $c.y$ is the lower midpoint along y-axis because $B.y > M.y$. Fig. 3.9 also shows another normalized half space $H'_{O:D}$ that also passes through the same point c.

Pruning Rule 3.3.1 *Let R_q and R_{fil} be the rectangular regions of the query q and a filtering object p, respectively. For any point p' that lies in $\bigcap_{i=1}^{4} H'_{C_i:C'_i}$, $mindist(p', R_q) > maxdist(p', R_{fil})$ where $H'_{C_i:C'_i}$ is normalized half space between C_i (the i^{th} corner of the rectangle R_{fil}) and its antipodal corner C'_i in R_q. Hence p' can be pruned.*

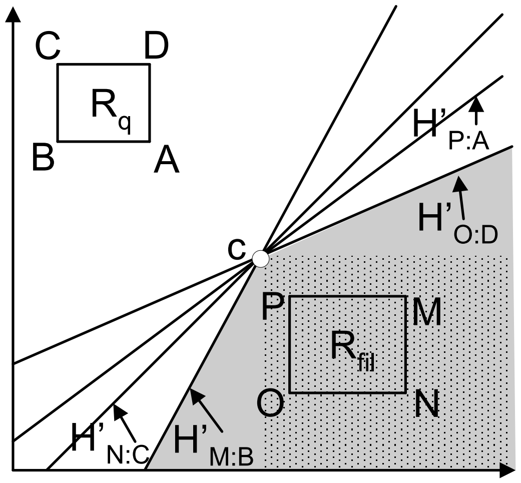


Figure 3.10: Half space pruning and dominance pruning

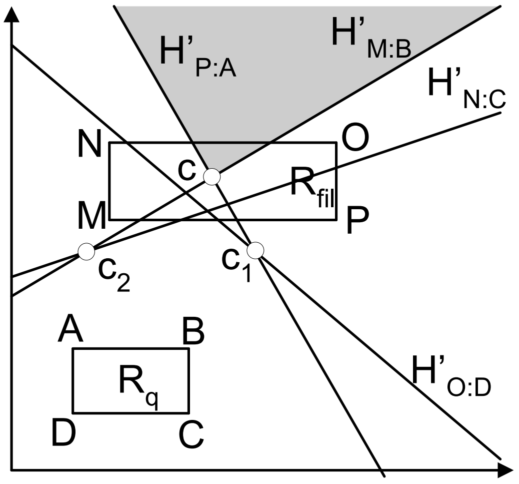


Figure 3.11: Any point in shaded area cannot be RNN of q

Fig. 3.10 shows an example of the half space pruning where the four normalized antipodal half spaces define the pruned region (the area shown shaded). The proof of correctness is non-trivial and is given in Appendix A (see Lemma A.2.5). Below, we present the intuitive justification of the proof.

Intuitively (as in the example of Fig. 3.8), if we draw all possible half spaces between all points of R_q and R_{fil} and move them to a point c for which $mindist(c, R_q) \geq maxdist(c, R_{fil})$, then the intersection of these half spaces correctly approximates the pruned region. Also note that in two dimensional space, at most two normalized spaces define such area. Consider the example of Fig. 3.10, where only $H'_{O:D}$ and $H'_{M:B}$ define the pruned region (the reason is that these two have largest and smallest slopes among all other possible half spaces). In fact, the antipodal corners are defined such that the half spaces having largest and smallest slopes are among the four antipodal half spaces. Moreover, the point c shown in Fig. 3.10 satisfies $mindist(c, R_q) = max\ dist(c, R_{fil})$ because normalized half spaces are defined such that c lies at the middle of the line that joins the corners A and N. Hence the four normalized antipodal half spaces correctly approximate the pruned region.

For ease of explanation, in Fig. 3.10, we have shown an example where the two rectangles R_q and R_{fil} do not overlap each other in any dimension. If the two rectangles overlap each other in any dimension (as in Fig. 3.11), the four half spaces do not meet at the same point. In Fig. 3.11, $H'_{O:D}$ and $H'_{P:A}$ are moved to c_1 and $H'_{N:C}$ and $H'_{M:B}$ are moved to point c_2. However, it can be verified by calculating the intersection that the half spaces that define the pruned region ($H'_{M:B}$ and $H'_{P:A}$) meet at a point c that satisfies $mindist(c, R_q) \geq maxdist(c, R_{fil})$.

Dominance Pruning

We first give the intuition behind this pruning rule. Consider the example of Fig. 3.10 again. The normalized half spaces are defined such that if R_{fil} and R_q do not overlap each other in any dimension then all the normalized antipodal half spaces meet at the same point c. This is because the point c is constructed using either the upper or the lower midpoint in each dimension depending on the x and y coordinate values of the two corners (see the definition of normalized half spaces and the four normalized half spaces in Fig. 3.10).

We also observe that the angle between the half spaces that define the pruned area (shown in grey) is always greater than 90°. Based on these observations, it can be verified that the space dominated by c (the dotted-shaded area) can be pruned. Formal proof is given in Appendix A (see Lemma A.2.6).

Let R_q be the rectangular region of q. We can obtain four regions as shown in Fig. 3.12. Let R_{fil} be the rectangular region of a filtering object that lies completely in one of the 4 regions. Let f be the furthest corner of R_{fil} from R_q and n be the nearest

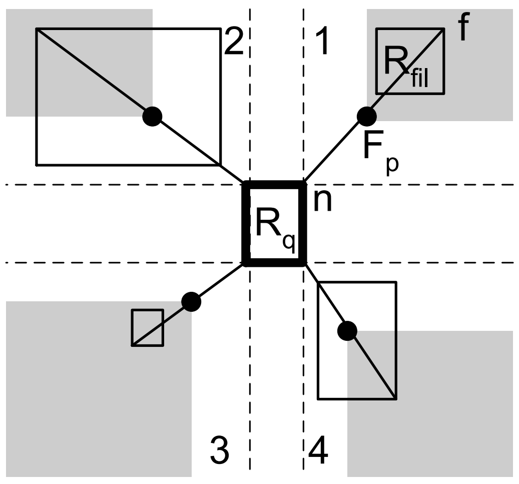


Figure 3.12: Shaded areas can be pruned

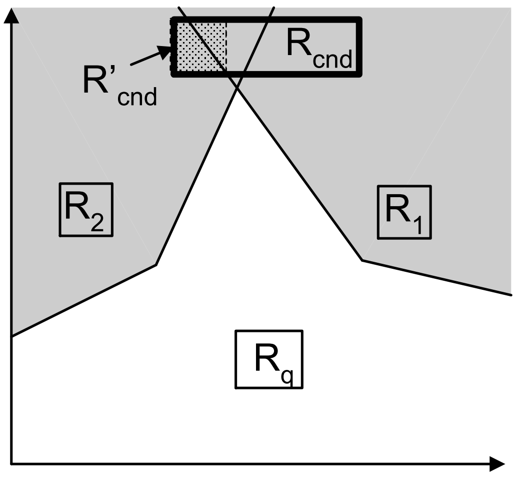


Figure 3.13: R_{cnd} can be pruned by R_1 and R_2

corner of R_q from f (as shown in region 1 of Fig. 3.12). A point F_p that lies at the centre of the line joining f and n is called a *frontier point*.

Pruning Rule 3.3.2 *Any candidate object p' that is dominated by the frontier point F_p of a filtering object cannot be RNN of q.*

Fig. 3.12 shows four examples of dominance pruning (one in each region). In each partition, the shaded area is dominated by the frontier point of that partition and can be pruned. Note that if R_{fil} overlaps R_q in any dimension, we cannot use this pruning rule because the normalized antipodal half spaces in this case do not meet at the same point. For example, the four normalized antipodal half spaces intersect at two points in Fig. 3.11. In general, the pruning power of this rule is less than that of the half space pruning. Fig. 3.10 shows the area pruned by the half space pruning (the shaded area) and dominance pruning (the dotted area). The main advantage of this pruning rule is that the pruning procedure is computationally more efficient than the half space pruning, as checking the dominance relationship is easier.

Metric Based Pruning

Pruning Rule 3.3.3 *A candidate object can be pruned if* $maxdist(R_{cnd}, R_{fil}) < mindist$ (R_{cnd}, R_q) *where* R_{cnd} *is the rectangular region of the candidate object.*

This pruning approach is the least expensive because it requires a simple distance comparison. Recall that the half space (or the dominance) pruning defines a region such that any point p' that lies in it is always closer to the filtering object than to q. Metric based pruning checks this by a simple distance comparison. However, this does not mean that the metric based pruning has at least as much pruning power as half space or dominance pruning. This is because the half space and dominance pruning can trim the rectangular region of a candidate object that lies in the pruned region. It may lead to pruning of a candidate object when more than one filtering objects are considered.

Consider the example of Fig. 3.13, where two rectangles R_1 and R_2 of two filtering objects are shown. The rectangle R_{cnd} cannot be pruned when half space pruning is applied on R_1 or R_2 alone. However, the rectangle R_{cnd} can be pruned when both R_1 and R_2 are considered. As in [TPL04], we use loose trimming of the rectangle by using trimming algorithm [GRSY97]. The trimming algorithm trims a part of the rectangle that cannot be pruned. First, R_{cnd} is pruned by the half spaces of R_1 and the trimming algorithm trims the rectangle that lies in the pruned region. The unpruned rectangle R'_{cnd} (shown with dotted shaded area) is returned. This remaining rectangle completely lies in the area pruned by R_2 so the candidate object is pruned. Note that metric based pruning cannot prune R_{cnd}.

Also note that if the exact location of a candidate object is known (R_{cnd} is a point) and metric based pruning fails to prune the object then half space pruning and dominance pruning also fail to prune the object. Hence, half space pruning and dominance pruning are applied only when the exact location of a candidate object is not known.

Pruning if exact location of query is known

If the exact location of the query or a filtering object is known, previous pruning rules can be applied by reducing the rectangles to points. However, a tighter pruning is possible if the exact location of the query is known. Below, we present a tighter pruning rule for such case.

Pruning Rule 3.3.4 *Let R_{fil} be a rectangle and q be a query point. For any point p that lies in $\bigcap_{i=1}^{4} H_{C_i:q}$ (C_i is the i^{th} corner of R_{fil}), $dist(p,q) > maxdist(p, R_{fil})$ and thus p cannot be the RNN of q.*

Proof Maximum distance between a rectangle R_{fil} and any point p is the maximum of distances between p and the four corners, i.e., $maxdist(p, R_{fil}) = max(dist(p, C_i))$ where C_i is the i^{th} corner of R_{fil}. Any point p that lies in a half space $H_{C_i:q}$ satisfies $dist(p,q) > dist(p, C_i)$ for the corner C_i of R_{fil}. Hence a point p lying in $\bigcap_{i=1}^{2d} H_{C_i:q}$, satisfies $dist(p,q) > maxdist(p, R_{fil})$. ∎

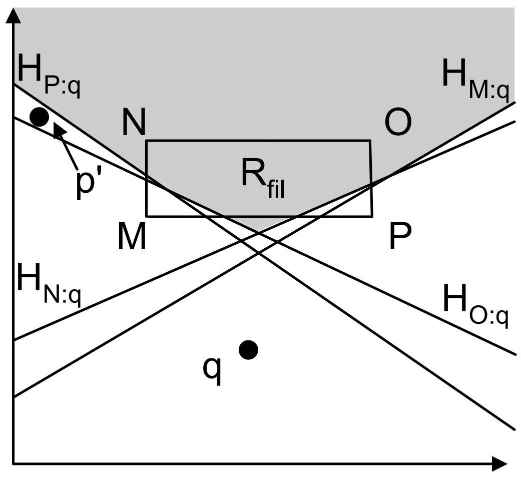


Figure 3.14: Half space pruning when exact location of query is known

Consider the example of Fig. 3.14 that shows the half spaces between q and the corners of R_{fil}. Any point that lies in the shaded area is closer to every point in rectangle R_{fil} than to q.

It is easy to prove that the pruned area is tight. In other words, any point p' that lies outside the shaded area may possibly be the RNN of q. Fig. 3.14 shows such point p'. Since it does not lie in $H_{P:q}$ it is closer to q than to the corner P. Hence it may be the RNN of q if the exact location of the filtering object is at corner P.

Algorithm 1 : Prune(R_q, S_{fil}, R_{cnd})

Input: R_q: rectangular region of q ; S_{fil}: a set of filtering objects ; R_{cnd}: the rectangular region of candidate object

Output: returns true if R_{cnd} is pruned; otherwise, returns false

Description:

```
1:  for each $R_{fil}$ in $S_{fil}$ do
2:    if $maxdist(R_{cnd}, R_{fil}) < mindist(R_q, R_{cnd})$ then /* Pruning rule 3.3.3 */
3:      return true
4:    if $mindist(R_{cnd}, R_{fil}) > maxdist(R_q, R_{cnd})$ then
5:      $S_{fil} = S_{fil} - R_{fil}$/* $R_{fil}$ cannot prune $R_{cnd}$ */
6:  if exact location of $cnd$ is known then
7:    return false/* the object cannot be pruned */
8:  for each $R_{fil}$ in $S_{fil}$ do
9:    if $R_{fil}$ is fully dominated by $R_q$ in a partition $P$ then /* Pruning rule 3.3.2 */
10:     trim the part of $R_{cnd}$ that is dominated by $F_p$
11:   return true if $R_{cnd}$ is pruned
12: return
13: for each $R_{fil}$ in $S_{fil}$ do
14:   Trim using half space pruning/* Pruning rule 3.3.1 */
15:   return true if $R_{cnd}$ is pruned
16: return false
```

Integrating the pruning rules

Algorithm 1 is the implementation of all the pruning rules. Specifically, we apply pruning rules in increasing order of their computational costs (i.e., metric based pruning, dominance pruning and then half space pruning). While simple pruning rules are not as restricting as more expensive ones, they can quickly discard many non-promising candidate objects and save the overall computational time.

Three subtle optimizations in the algorithm are:

1. As stated in Section 3.3.2, if the exact location of the candidate object is known then only metric based pruning is required. So, we do not consider dominance and half space pruning for such candidates (line 7).

2. If $mindist(R_{cnd}, R_{fil}) > maxdist(R_q, R_{cnd})$ for a given MBR R_{fil}, then R_{fil} cannot prune any part of R_{cnd}. Hence such R_{fil} is not considered for dominance and half space pruning (lines 4-5).

3. If the frontier point F_{p_1} of a filtering object R_{fil_1} is dominated by the frontier point F_{p_2} of another filtering object R_{fil_2}, then F_{p_1} can be removed from S_{fil} because the area pruned by F_{p_1} can also be pruned by F_{p_2}. However, note that a frontier point cannot be used to prune its own rectangle. Therefore, before deleting F_{p_1}, we use it to prune the rectangle belonging to F_{p_2}. This optimization reduces the cost of dominance pruning. To maintain the simplicity, we do not show this optimization in Algorithm 1.

3.3.3 Continuous RNN Monitoring

Data Structure

Our system has an object table and a query table. Object table (query table) stores the id and the rectangular region for each object (query). In addition, the query table stores a set of candidate objects S_{cnd} for each query.

Main-memory computation is the main paradigm in on-line/real-time query processing [MHP05, KMS$^+$07, XZ06]. Grid structure is preferred when updates are intensive [MHP05] because complex data structures (e.g., R-tree, Quad-tree) are expensive to update. For this reason, we choose grid-based data structure to store the locations and rectangular regions of moving objects and queries. Each cell contains two lists: 1) *object list*; 2) *influence list*. Object list of a cell c contains object id of every object whose rectangular region overlaps the cell c. This list is used to identify the objects that may be located in this cell. Influence list of a cell c contains query ids of all queries for which this cell lies in (or overlaps with) the unpruned region. The intuition is that if an object moves into this cell, we know that the queries in the influence list of this cell are affected.

Range queries and constrained NN queries (nearest neighbors in constrained region) are issued to compute RNNs of a query (e.g., six constrained nearest neighbor queries are issued in the six-regions based approach). In our algorithm, we also need an algorithm to search the nearby objects in a constrained area (the unpruned region). Several continuous nearest neighbors algorithms [YPK05, MHP05, XMA05] based on grid-based index have been proposed. However, the extension of these grid-access methods for queries on constrained area becomes inefficient. i.e., the cells around queries are retrieved even if they lie in the pruned region. To efficiently search nearest neighbors in a constrained area, we introduce *conceptual grid tree*.

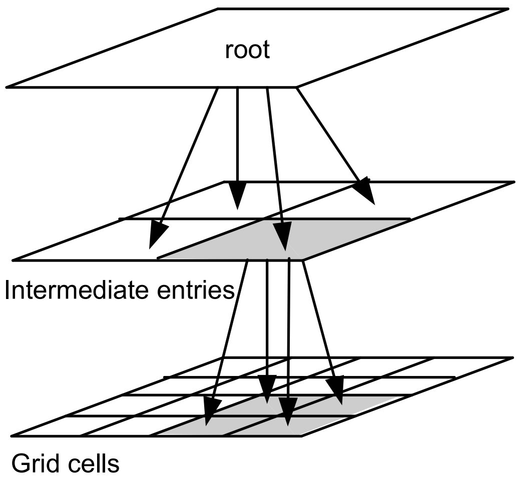


Figure 3.15: Conceptual grid-tree of a 4× 4 grid

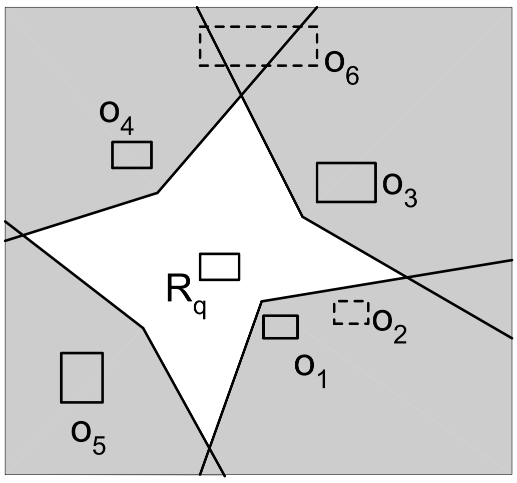


Figure 3.16: Illustration of the filtering phase

Fig. 3.15 shows an example of the conceptual grid-tree of a 4×4 grid. For a grid-based structure containing $2^n \times 2^n$ cells where $n \geq 0$, the root of our conceptual grid-tree is a rectangle that contains all $2^n \times 2^n$ cells. Each entry at l-th level of this grid-tree contains $2^{(n-l)} \times 2^{(n-l)}$ cells (root being at level 0). An entry at l-th level is divided into four equal non-overlapping rectangles such that each such rectangle contains $2^{(n-l-1)} \times 2^{(n-l-1)}$ cells. Any n-th level entry of the tree corresponds to one cell of the grid structure. Fig. 3.15 shows root entry, intermediate entries and the cells of grid. Note that the grid-tree does not exist physically, it is just a conceptual visualisation of the grid.

The spatial queries algorithms that can be applied on R-tree can easily be applied on the conceptual grid tree. The advantage of using this grid-tree over previously used grid-based access methods is that if an intermediate entry of the tree lies in the pruned region, none of the cells inside it are accessed.

Initial Computation

The initial computation consists of two phases namely *filtering* and *verification*. Below we discuss them in detail.

Filtering

In this phase (Algorithm 2), the grid-tree is traversed to select the candidate objects and these objects are stored in S_{cnd}. These candidate objects are also used to prune other objects. Initially, root entry of the grid-tree is inserted in a min-heap H. We try to prune every de-heaped entry e (line 6) by using the pruning rules presented in the previous section. If e is a cell and cannot be pruned, we insert the objects into heap that are in its object list. Otherwise, if e is an intermediate entry of the grid-tree, we insert its four children into the heap H with key $mindist(c, R_q)$. If e is an object and is not pruned, we insert it into S_{cnd}. The algorithm stops when the heap becomes empty.

Fig. 3.16 shows an example of the filtering phase. For better illustration, the grid is not shown. Objects are numbered in order of their proximity to q. Algorithm iteratively finds the nearest objects and prunes the space accordingly. In the example of Fig. 3.16,

Algorithm 2 : Filtering

```
1:  for each query $q$ do
2:    $S_{cnd} = \phi$
3:    Initialize a min-heap $H$ with root entry of Grid-Tree
4:    while H is not empty do
5:      de-heap an entry $e$
6:      if (not Pruned($R_q, S_{cnd}, e$)) then /* Algorithm 1 */
7:        if $e$ is a cell in Grid then
8:          for each object $o$ in object list of $e$ do
9:            insert $o$ into H if not already inserted
10:       else if $e$ is an intermediate entry of grid-tree then
11:         for each of its four children $c$ do
12:           insert $c$ into $H$ with key $mindist(c, R_q)$
13:       else if $e$ is an object then
14:         $S_{cnd} = S_{cnd} \cup \{e\}$
```

the algorithm first finds o_1 and prunes the space. Since the next closest object o_2 lies in the pruned space, it is not considered and o_3 is selected instead. The algorithm continues and retrieves o_4 and o_5 and the shaded area is pruned. The algorithm stops because there is no other object in the unpruned area (the white area). The rectangles of the pruned objects are shown in broken lines.

One important note is that in this phase, the call to pruning algorithm at line 6 does not consider the exact locations of any object or query for pruning even if the exact location is known. This is because we want to find a set of candidate objects S_{cnd} such that as long as all of them remain in their rectangular regions and no other object enters in the unpruned area, the set of candidate objects is not affected. For example, the set of candidate objects $\{o_1, o_3, o_4, o_5\}$ will not change unless q or any candidate object moves out of its rectangular region or any of the remaining objects (o_2 and o_6) moves in the unpruned area (the white area).

Marking the cells in unpruned area: To quickly identify that an object has moved into the unpruned area of a query q, each cell that lies in the unpruned area is marked. More specifically, q is added in the influence list of such cell. We mark these cells in a hierarchical way by using the grid-tree. For example, if an entry completely lies in the unpruned region, all the cells contained by it are marked. The cells are unmarked similarly.

Verification

At this stage, we have a set of candidate objects S_{cnd} for each query. Now, we proceed to verify the objects. Since every query q reports its location to the server at every timestamp, we can use its location to further refine its S_{cnd}. More specifically, any object $o \in S_{cnd}$ cannot be the RNN of q for which $mindist(o, q) \geq maxdist(o, o')$ for any other $o' \in S_{cnd}$. If the object cannot be pruned by this distance based pruning, we try to prune it by using pruning rule 3.3.4. For every query q, its candidate objects that cannot be pruned are stored in a list S_{global}.

The server sends messages to every object in S_{global} for which the exact location is not known. The objects send their exact locations in response. For each query q, the list of candidate objects is further refined by using these exact locations. As noted in [SAA00], at this stage, the number of candidate objects for a query cannot be greater than six in two dimensional space. We verify these candidate objects as follows.

Algorithm 3 : Verification

```
1: Refine $S_{cnd}$ using the exact location of $q$
2: Request objects in $S_{cnd}$ to send their exact locations
3: Select candidate objects based on exact location of the objects
4: Verify candidate objects (at most six) by issuing boolean range queries
```

For a candidate object o, we issue a *boolean range query* [SFT03] centered at o with range $dist(o, q)$. In contrast to the conventional range queries, a boolean range query does not return all the objects in the range. It returns true if an object is found within the range, otherwise it returns false. Fig. 3.17 shows an example, where candidate objects are o_1 to o_4. Any object for which its exact location in its rectangular region is not known is shown as a shaded rectangle (see objects o_6, o_7 and o_8). The rectangular regions of the objects for which we know the exact locations are shown in dotted rectangles (see objects o_1 to o_5 and the query q).

The object o_3 cannot be the RNN because o_5 (for which we know the exact location) is found within the range. Similarly, o_4 cannot be the RNN because the rectangular region of o_6 completely lies within the range. The object o_2 is confirmed as RNN because no object is found within the range. The only candidate object for which the result is undecided is o_1 because we do not know the exact location of object o_8 which may or may not lie within the range. The server needs its exact location in order to verify o_1. For each query q, the server collects all such objects. Then, it sends messages to all these objects and verifies all undecided candidate objects upon receiving the exact locations.

Continuous Monitoring

The set of candidate objects S_{cnd} of a query changes only when the query or one of the candidate objects leaves its rectangular region or when any other object enters into the unpruned region. If S_{cnd} is not affected, we simply call the verification phase to update the results. Otherwise, we have to update S_{cnd}.

Consider the running example of Fig. 3.17 that shows a query q and its four candidates (o_1 to o_4). Assume that after several timestamps, one of the candidate objects (see o_1 in Fig. 3.18) moves out of its rectangular region. We need to call the filtering phase again because the pruned region is not valid anymore and S_{cnd} may have changed.

One possible approach to update S_{cnd} is to call the filtering phase (Algorithm 2) from scratch. Second possible approach to update S_{cnd} is to call Algorithm 2 with S_{cnd} set to $\{o_2, o_3, o_4\}$ instead of initializing an empty S_{cnd}. Note that the object that moves out of its rectangular region (e.g., o_1) has not been considered in S_{cnd}. If it is still the candidate object it will be retrieved during the execution of Algorithm 2. In our initial experiments, we found that the second approach to update S_{cnd} is almost as expensive

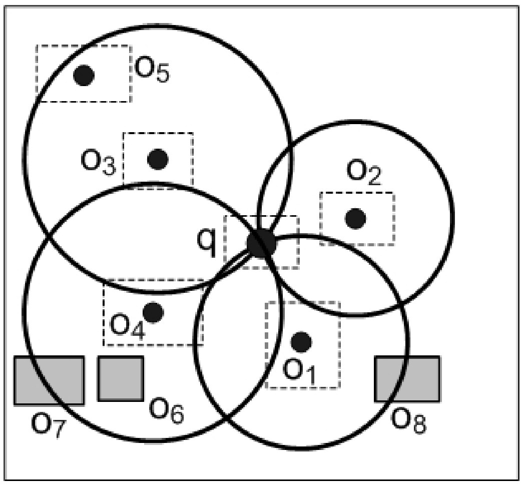


Figure 3.17: Verification phase

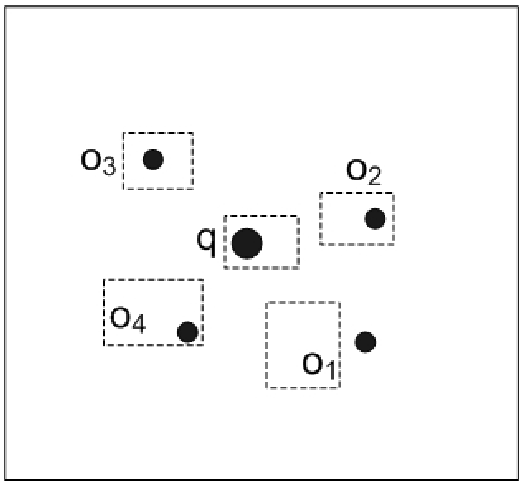


Figure 3.18: Continuous monitoring

as the first approach. Below, we show that if we choose to compute S_{cnd} from scratch, we may save computation cost in upcoming timestamps.

Consider the example of Fig. 3.18 where the candidate object o_1 leaves its rectangular region. Since the query and other candidate objects are also moving, they are likely to leave their regions in next few timestamps which will trigger the expensive filtering phase again. For example, it is possible that the object o_4 leaves its rectangular region in the next timestamp and we have to call the expensive filtering phase again. To overcome this problem, we request all the candidate objects to send their exact locations as well as their new rectangular regions (note that this does not increase the communication cost because in any case we need to contact these candidate objects in the verification phase at line 2 of Algorithm 3). After receiving these new rectangular regions, we update S_{cnd} by calling the filtering phase from scratch. Now the candidate objects have new rectangular regions and they are expected to remain in their respective rectangular regions for longer.

Suppose that an object o is a candidate for two queries q_1 and q_2 and S_{cnd} of q_1 is affected by a location update of any other object o'. We cannot ask o to update its rectangular region because it will affect S_{cnd} of query q_2 as well. Hence, the server only asks an object to update its rectangular region if it does not affect other queries.

3.3.4 Cost Analysis

In this section, we analyse the computation and communication cost for our proposed solution. First, we present a pruning rule based on six-regions approach and compute the communication cost. Then, we show that the pruning rules used in our technique are superior. Hence the communication cost gives an upper bound. Then, we analyse the computation cost.

Assumptions: We assume that the system contains N objects in a unit space (extent of the space on both dimensions is from 0 to 1). Each rectangular region is a square and width of each side is w. The centers of all rectangular regions are uniformly distributed.

Communication cost: Consider the example of Fig. 3.19 where a 60° region bounded by the angle $\angle EqC$ is shown in thick lines. Suppose that we find a filtering object whose rectangular region R_{fil} is fully contained in the region. Any object o' can be pruned if $dist(o', q) \geq maxdist(R_{fil}, q)$. In other words, the possible candidates may lie only in the space defined by qEC where EC is an arc and $qC = qE = maxdist(R_{fil}, q)$.

Let r be the distance between q and the center of R_{fil}. Then, $maxdist(R_{fil}, q) \leq r + w/\sqrt{2}$ where $w/\sqrt{2}$ is the half of the diagonal length of R_{fil}. Since, all objects are represented by rectangular regions, any object is possible RNN candidate that has its centre at a distance not greater than $w/\sqrt{2}$ from the region qEC. So, the range becomes $(r + \sqrt{2}w)$. Total number of candidates that overlap or lie within the region qEC is

$$\frac{\pi(r + \sqrt{2}w)^2 N}{6}$$

Let R be the maximum of r of all six regions, the total number of candidate objects is bounded by

$$| S_{cnd} | = \pi(R + \sqrt{2}w)^2 N \tag{3.1}$$

The server sends request to all these candidate objects and receives their exact locations. So the total number of messages M_1 at this stage is bounded by

$$M_1 = 2\pi(R + \sqrt{2}w)^2 N \tag{3.2}$$

After receiving the updates, the server eliminates the candidate objects that cannot be the RNN (based on their exact locations). As proved in [SAA00], the number of candidate objects cannot be greater than six. Hence, the server needs to verify those six candidate objects. In order to verify a candidate object o, the server issues a range query of distance $dist(o, q)$ centered at o. In worst case, all the objects that lie within this range must report their exact locations. Total number of objects that overlap or lie within the range is

$$\pi(dist(o, q) + w/\sqrt{2})^2 N$$

Since these candidate objects belong to the nearest neighbors in each region, $dist(o, q)$ corresponds to the distance of closest object in the region. For all six regions, the maximum of $dist(o, q)$ is the distance of sixth nearest neighbor from q (assuming uniform distribution). So the maximum range is the radius of a circle around q that contains six objects. As we assume a unit space, the radius of such circle that contains six objects is $\sqrt{\frac{6}{N\pi}}$. So the maximum number of messages M_2 required to verify all six candidate objects is

$$M_2 = 6 \times 2\pi(\sqrt{\frac{6}{N\pi}} + w/\sqrt{2})^2 N$$

$M_1 + M_2$ are the messages required to retrieve the *server-initiated* updates. Let M_3 be the number of *source-initiated* updates (the objects that leave their rectangular regions). Let v be the average speed of objects. An object starting at center of the square of width w and moving with speed v will take at least $w/2v$ time to leave the region. So, total number of updates M_3 at each timestamp is

$$M_3 = N \times min(\frac{2v}{w}, 1)$$

Note that the equation bounds the number of source-initiated updates by N. The total communication cost per timestamp is $(M_1 + M_2 + M_3 + 1)$ where 1 denotes the location update of the query. Note that if w is small, the number of source-initiated updates M_3 increases and if w is large, the number of server-initiated updates $(M_1 + M_2)$ increases.

Now, we find R. Note that to use the pruning of Fig. 3.19, we had assumed that R_{fil} completely lies in the 60 degree region EqC. Hence r in Equation (3.1) corresponds to the distance of the closest object in each region that completely lies in it. Similarly, R is the maximum of r of each region.

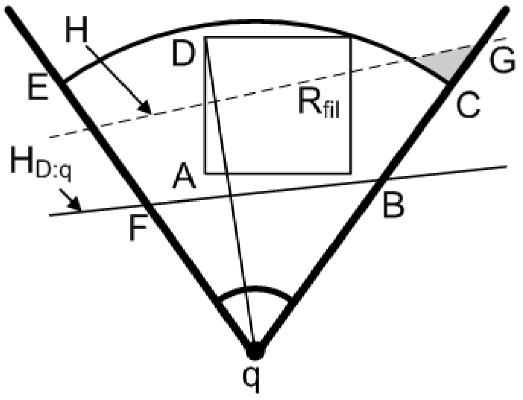


Figure 3.19: Half space pruning vs six-regions based pruning

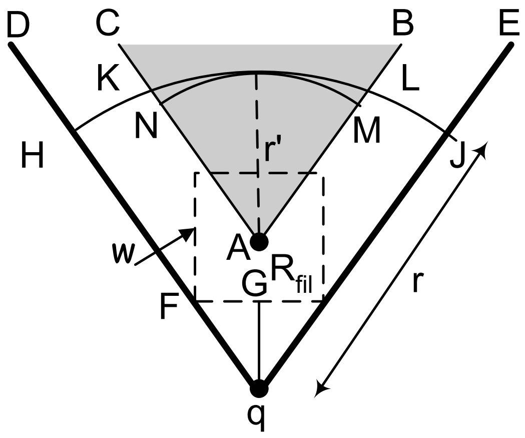


Figure 3.20: An object completely lying in the 60° region

Fig. 3.20 shows a region DqE and a rectangular region R_{fil} of a filtering object (shown in broken line). Note that any rectangular region of side length w with center lying in ABC (the shaded area) will completely lie in the region DqE. In other words, r corresponds to the closest object of q in the region that has center lying in ABC.

Let $r = qH = qJ$ as shown in Fig. 3.20. Let the radius belonging to area AMN be r'. The radius r' can be computed as $r' = r - qA$ where $qA = qG + GA = qG + w/2$. The length of $qG = 0.866w$ which can be found by the triangle FGq where $FG = w/2$ and $\angle GFq = 60°$. Hence $r' = r - 1.366w$.

It can be verified that when $r = \sqrt{\frac{6}{N\pi}} + 1.366w$, then $\pi(r')^2N = 6$. In other words when radius is r, one object in each region will be found such that it completely lies in the region. So M_1 can be rewritten as

$$M_1 = 2\pi(\sqrt{\frac{6}{N\pi}} + 2.78w)^2 N$$

The cost $(M_1 + M_2 + M_3 + 1)$ is the cost for one RNN query. The cost of multiple RNN queries is $\mid Q \mid \cdot (M_1 + M_2 + 1) + M_3$ where $\mid Q \mid$ is the number of queries.

Now, we show that the area pruned by our proposed approach (pruning rule 3.3.4) contains the area pruned by previously described six regions based approach. Consider the example of Fig. 3.19 where R_{fil} completely lies in the region. The area pruned by six-regions approach is the area of region outside qCE where CE is an arc and $qC = maxdist(R_{fil}, q)$. Our pruning approach prunes the area defined by the intersection of the four half spaces between q and the corners of R_{fil}. Fig. 3.19 shows a half space H (shown in broken line) that crosses the region at a point G such that $qG > qC$. This half space fails to prune some area pruned by the six region based approach (the six region based approach prunes the shaded area which this half space H fails to prune).

In order to prove that our pruning approach always contains the area pruned by the six-region based approach, we need to show that all four half spaces between q and the corners of R_{fil} cross the region at a point B such that $qB \leq qC$. Fig. 3.19 shows a half space $H_{D:q}$ between corner D and q. Consider the right triangle qAB where $\angle BqA \leq 60°$. The length of qB is $\frac{qA}{cos(\angle BqA)}$. The maximum possible value of qB is $2 \times qA$ when $\angle BqA$ is $60°$. Since $2 \times qA = qD$ and $qD \leq qC = maxdist(R_{fil}, q)$, so $qB \leq qC$. Similarly, it can be proved that $qF \leq qE$. Hence all the four half spaces contain the area pruned by the region based approach.

Computation cost: Let C_{fil} and C_{ver} be the costs of the filtering phase and the verification phase, respectively. The computation cost at each timestamp is $\rho \times C_{fil} + C_{ver}$ where ρ is the probability that at a given timestamp at least one of the following two events happens: i) the query or any of the candidate objects leaves its safe region; ii) any other object enters in the unpruned region of the query.

The verification cost includes using the exact locations of M_1 objects to further refine the set of candidate objects and using boolean range queries to verify the remaining candidate objects (at most six). Let the cost of refining an object be C_{ref} and the cost of a boolean range query be C_{br}, the verification cost is $C_{ver} = M_1 \times C_{ref} + \mid S_{cnd} \mid \times C_{br}$ where $\mid S_{cnd} \mid \leq 6$.

3.3.5 Extensions

Since our proposed pruning rules can be applied in multidimensional space, the extension of our algorithm to arbitrary dimensionality is straightforward. Below, we present extension of our algorithm to RkNN monitoring.

RkNN Pruning: An object cannot be RkNN of a query if it is pruned by at least k filtering objects. We initialize a counter to zero and trim R_{cnd} by each filtering object. When the whole rectangle is trimmed, the counter is incremented and the original rectangle is restored. We continue this process by trimming with remaining filtering objects. If the counter becomes equal to k, the object is pruned.

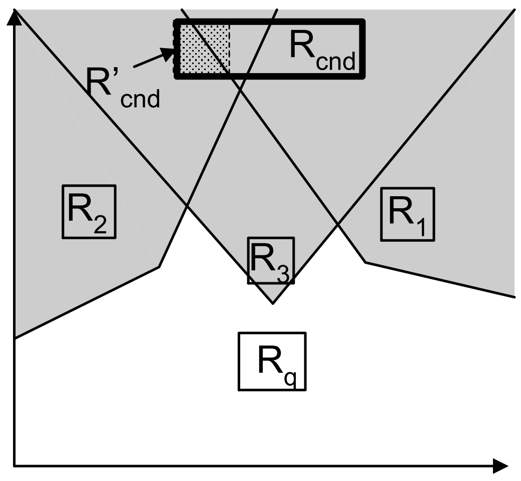


Figure 3.21: RkNN Pruning

Suppose k is 2 and consider the example of Fig. 3.21 where R_{cnd} and three filtering objects R_1, R_2 and R_3 are shown. Filtering objects are considered in order R_1, R_2 and R_3. R_{cnd} is trimmed to R'_{cnd} when R_1 is used for pruning. R'_{cnd} is completely pruned by R_2. The counter is incremented to one and the original rectangle R_{cnd} is restored. Now,

R_{cnd} is trimmed by R_3 and the counter is incremented to two because whole rectangle is trimmed. The algorithm prunes R_{cnd} because it has been pruned two times.

Note that if the filtering objects are processed in order R_1, R_3 and R_2, the candidate object cannot be pruned. Finding the optimal order is difficult and trying all possible orders is computationally expensive. This will make filtering of this candidate object more expensive than its verification. Hence, if a candidate object is not pruned by the above mentioned pruning, we consider it for verification.

RkNN Verification: An object o cannot be RkNN if the range query centered at o with range $dist(o, q)$ contains greater than or equal to k objects. Otherwise, the object is reported as RkNN. Suppose k is 2 and consider the example of Fig. 3.17 again. The candidate objects o_2 and o_3 are confirmed as R2NNs because there are less than 2 objects within their ranges. The object o_1 is also confirmed because at most one object (o_5) lies within the range. The result for o_4 is undecided, so the location of o_7 is requested. Note that we do not need to request the exact location of o_6.

Bichromatic Queries: Now, we briefly present the extension of our proposed solution to bichromatic queries. Let there be two sets of objects O and P and query q belongs to O. The area is pruned by iteratively finding nearby filtering objects that belong to O and lie in the unpruned region. The pruning of area is stopped when there is no filtering object in the unpruned region. The objects of type P that lie in the unpruned region are the candidate objects. The server asks these candidate objects to report their exact locations. Upon receiving the exact locations, any candidate object p is reported as RNN if there does not lie an object of type O within a circle with radius $dist(p, q)$ centered at p. If the result is undecided, type O objects that have rectangles overlapping with the circles are requested to send their locations. Based on these received locations, the result is computed and reported to the client.

3.4 Query Processing in Spatial Networks

In this section, we present our technique to continuously monitor RNN queries in spatial networks. First, we introduce the basic concepts and notations in Section 3.4.1. In Section 3.4.2, we study the problem characteristics. Section 3.4.3 presents the framework of our technique. Filtering and verification techniques are presented in Section 3.4.4 and Section 3.4.5, respectively. We present the extensions of our RNN monitoring algorithm to other variants of RNN queries in Section 3.4.7.

3.4.1 Terminology

First we define few terms and notations.

Spatial network G is a weighted graph consisting of *vertices* and *edges*. An edge between two vertices v_1 and v_2 is denoted as $e(v_1, v_2)$. Each edge has a positive weight that denotes the cost of travelling on that edge (e.g., length of the edge, time taken to travel along the edge etc.). The weight of an edge $e(v_1, v_2)$ is denoted as $|e(v_1, v_2)|$.

Segment $s_{[x,y]}$ is the part of an edge between x and y where both x and y are points on the edge. By definition, an edge is also a segment defined by the end points (vertices) of the edge. The weight of a segment $s_{[x,y]}$ is denoted as $|s_{[x,y]}|$.

Fig. 3.22 shows an example of a road network with eight vertices (a to h). Six objects (o_1 to o_5 and q) are also shown. The query object q is shown as a black star. Several segments are also shown. For instance, the edge $e(b, g)$ consists of segments $s_{[b,o_5]}$, $s_{[o_5,o_4]}$, $s_{[o_4,m]}$, $s_{[m,o_3]}$ and $s_{[o_3,g]}$. The weights of edges and segments are also shown. For example, the weight of the edge $e(c, g)$ is 5 and the weight of the edge $e(b, g)$ is $2+4+2+2+2 = 12$.

Shortest network distance $SNDist(x, y)$ between any two points x and y is the minimum network distance between x and y (i.e., total weight of the edges on the shortest path from x to y). In Fig. 3.22, the shortest path from q to o_4 is $q \rightarrow c \rightarrow g \rightarrow o_4$ and $SNDist(q, o_4)$ is 14.

In Section 3.2.1, we had formally defined the RNN queries based on the distance function $dist()$. In spatial networks, the RNN query uses the distance function such that it returns the shortest network distance between the points (i.e, $dist(x, y) = SNDist(x, y)$).

3.4.2 Problem Characteristics

In this section, we study the problem characteristics. The lemma below identifies the objects that cannot be the RNN of a query q.

Lemma 3.4.1 *An object o cannot be the RNN of q if the shortest path between q and o contains any other object o'.*

Proof If an object o' lies on the shortest path between q and o, this implies that $SNDist(o, o') < SNDist(o, q)$. Hence o is not the RNN of q. ∎

In Fig. 3.22, the object o_4 is not the RNN of q because the shortest path from q to o_4 is $q \rightarrow c \rightarrow g \rightarrow o_4$ which contains another object o_3.

Before we present next lemma, we define *dead* vertices. A vertex v is called a dead vertex if there exists an object o such that $SNDist(v, o) < SNDist(v, q)$. The object o is called the killer object of v because this is the object that makes the vertex v a dead vertex. In Fig. 3.22, the vertex g is a dead vertex and o_3 is its killer object. The vertex a is not a dead vertex. Note that a dead vertex may have more than one killer objects. For example, o_3, o_4 and o_1 are the killer objects of the vertex g.

Lemma 3.4.2 *An object o cannot be the RNN of q if the shortest path between q and o contains a dead vertex v with a killer object o' where $o' \neq o$.*

Proof Assume that a dead vertex v exists on the shortest path between q and o. The shortest network distance between o and q is $SNDist(o, q) = SNDist(o, v) + SNDist(v, q)$. Let o' be the killer object of vertex v. The shortest network distance between o and o' is $SNDist(o, o') \leq SNDist(o, v) + SNDist(v, o')$. By definition of a dead vertex v, $SNDist(v, o') < SNDist(v, q)$. Hence, $SNDist(o, o') < (SNDist(o, v) + SNDist(v, q) = SNDist(o, q))$. Hence, o cannot be the RNN of q. ∎

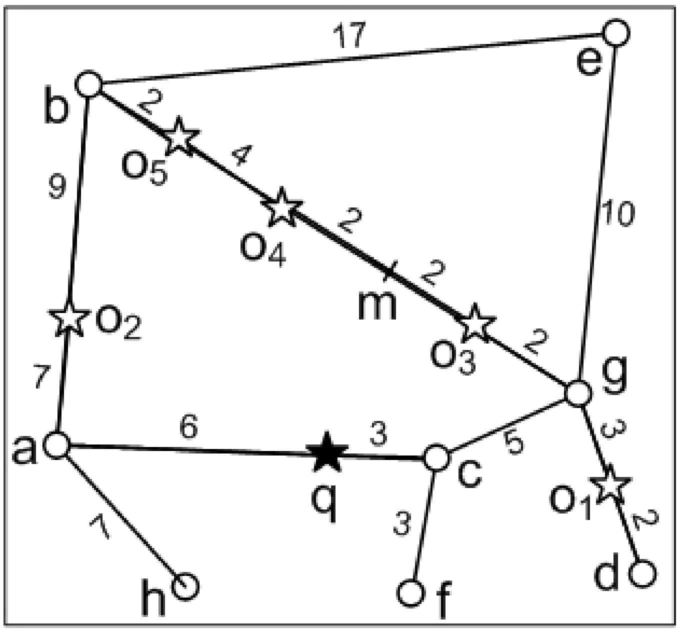


Figure 3.22: RNN query in a spatial network

In Fig. 3.22, the shortest path from q to o_3 contains a dead vertex g with a killer object o_1. The object o_3 is not the RNN of the query q because $SNDist(o_1, o_3) < SNDist(q, o_3)$. Similarly, the object o_1 is also not the RNN because the shortest path between q and o_1 contains the dead vertex g with a killer object o_3.

Lemma 3.4.3 *A vertex v' is a dead vertex if the shortest path between q and v' contains a dead vertex v.*

Proof Let o be the killer object of the vertex v. Then, $SNDist(v', o) \leq SNDist(v', v) + SNDist(v, o)$. By definition of a dead vertex v, $SNDist(v, o) < SNDist(v, q)$. Hence, $SNDist(v', o) \leq (SNDist(v', v) + SNDist(v, q) = SNDist(v', q))$. Hence v' is a dead vertex. ∎

In Fig. 3.22, the shortest path from q to e is $q \rightarrow c \rightarrow g \rightarrow e$ which contains a dead vertex g. Hence, e is also a dead vertex.

Lemma 3.4.4 *Consider an edge $e(v_1, v_2)$ that contains at least two objects on it and assume that the query q does not lie on it. The edge cannot contain any RNN if*

$SNDist(v_1, q) \geq |e(v_1, v_2)|$ and $SNDist(v_2, q) \geq |e(v_1, v_2)|$ where $|e(v_1, v_2)|$ is the weight of the edge.

Proof $|e(v_1, v_2)| \geq SNDist(o, o')$ for any two objects o and o' that lie on the edge $e(v_1, v_2)$. For any object o on the edge $e(v_1, v_2)$, the shortest path between o and q either passes through v_1 or v_2. Hence, $SNDist(o, q) \geq min(SNDist(v_1, q), SNDist(v_2, q)) \geq |e(v_1, v_2)| \geq SNDist(o, o')$. Hence, o cannot be the RNN of q. ∎

Before we present next lemma, we define extreme objects of an edge. An object o is called an extreme object of an edge $e(v_1, v_2)$ if either the segment $s_{[o,v_1]}$ or the segment $s_{[o,v_2]}$ does not contain any other object o'. In Fig. 3.22, the objects o_3 is an extreme object of the edge $e(b, g)$ because the segment $s_{[o_3,g]}$ does not contain any other object. Similarly, the object o_5 is also an extreme object because the segment $s_{[o_5,b]}$ does not contain any other object. However, the object o_4 is not an extreme object because both the segments $s_{[o_4,b]}$ and $s_{[o_4,g]}$ contain an object other than o_4. By definition of extreme objects, each edge contains at most two extreme objects. This holds true even if more than one objects lie at the same location. For instance, in Fig. 3.22, if there was an object o' at the same location as of o_3 then both o_3 and o' would not be the extreme objects.

Lemma 3.4.5 *Only the extreme objects of an edge can be the RNN of a query q given that q does not lie on the edge.*

Proof Let o be an object on the edge $e(v_1, v_2)$ and o be not an extreme object. Since q does not lie on the edge $e(v_1, v_2)$, the shortest path between o and q either passes through v_1 or v_2. Since o is not an extreme object, each of the segment $s_{[o,v_1]}$ and $s_{[o,v_2]}$ contains at least one object other than o. Hence, the shortest path from o to q contains at least one other object and o cannot be the RNN of q as implied by Lemma 3.4.1. ∎

In Fig. 3.22, the object o_4 cannot be the RNN of q because it is not an extreme object.

Lemma 3.4.6 *Regardless of the number of queries in the system, an edge that does not contain any query has at most two objects that can be the RNNs of any of the queries.*

Proof From Lemma 3.4.5, only the extreme objects can be the RNN of a query q. Since each edge contains at most two extreme objects, only at most two objects can be the RNNs of any of the queries. ∎

In Fig. 3.22, assume that the object o_2 is also a query point. Only the extreme objects (o_3 and o_5) of the edge $e(b, g)$ can be the RNNs of the query points q and o_2. Lemma 3.4.5 and Lemma 3.4.6 imply that the extreme objects of an edge are the only possible candidate objects for the queries that do not lie on the edge. Moreover, several queries may share same candidate objects.

Based on the problem characteristics we studied in this section, we develop an algorithm to continuously monitor RNN queries. The next section presents the framework of our proposed technique.

3.4.3 Framework

To simplify the presentation, we assume that the safe regions of the objects and queries are segments. Later in Section 3.4.6, we show that our technique can support the safe regions that consist of more than one edges and segments.

Each object and query is assigned a segment that is its safe region. The safe region of an object o is denoted as $o.s_{[x,y]}$. Since the safe region of an object is a segment, we use $o.x$ and $o.y$ to denote the end points of this segment. Each object and query reports its location to the server whenever it leaves its safe region. Such updates are called *source-initiated* updates. In order to update the results, the server might need to know the exact locations of some objects. The server receives the exact location of each such object by requesting its current location. Such updates are called *server-initiated* updates.

The safe region of a query q is chosen such that $q.s_{[x,y]}$ does not overlap with the safe region of any other object. The segment $q.s_{[x,y]}$ is considered as an edge and the end points $q.x$ and $q.y$ are considered as vertices. This is to simplify the presentation because Lemma 3.4.5 and Lemma 3.4.6 are applicable to every edge if the segment that contains the query is considered as a different edge.

The continuous monitoring algorithm consists of two phases.

1. Filtering. In filtering phase, the set of candidate objects are retrieved by pruning the objects that cannot be the RNN of a query q. The edges and segments of the network that cannot contain any RNN are also pruned. The part of the network that is not pruned is called unpruned network. The set of candidate objects remain valid unless at least one of the following happens: i) the query or a candidate object leaves its safe region; ii) an object enters in the unpruned network. Hence, the filtering phase is called only when at least one of the above two happens.

2. Verification. In verification phase, for each candidate object, the server checks if the candidate object is the RNN of q or not. More specifically, if q is the closest object of o (in terms of $SNDist$), the object o is reported as the RNN. The verification phase is called at each timestamp.

In the following, we present the details of both the filtering and verification phases.

3.4.4 Filtering

The main idea is to incrementally expand the network around the query in a way similar to Dijkstra's algorithm. More specifically, the vertices are accessed in increasing order of their $SNDist$ from q (a min-heap is used). Whenever a vertex v is de-heaped, its adjacent vertices are inserted in the heap if v is not a dead vertex. Lemma 3.4.1 and Lemma 3.4.2 are used to identify the candidate objects lying on the adjacent edges of v. The algorithm stops when the heap becomes empty.

Algorithm 4 presents the details. The set of candidate objects is S_{cnd} and is initialized to an empty set. Let $q.s_{[x,y]}$ be the safe region of the query. As mentioned earlier, the end points of the safe regions of the queries are treated as the vertices. A min-heap is initialized and $q.x$ and $q.y$ are inserted with keys set to zero. The entries from the heap are retrieved iteratively (line 5). When a vertex v is de-heaped, we consider its adjacent vertices iteratively (line 6). Let v' be an adjacent vertex of v. We obtain an object o

Algorithm 4 : Filtering

```
1: $S_{cnd} = \phi$
2: initialize a min-heap $H$
3: insert $q.x$ and $q.y$ with keys set to zero
4: while $H$ is not empty do
5:   deheap a vertex $v$ from $H$ and mark it as visited
6:   for each unvisited adjacent vertex $v'$ of $v$ do
7:     if there exists at least one object on $e(v, v')$ then
8:       get the object $o$ closest to $v$
9:       Assign $o$ a safe region $o.s_{[x,y]}$ and insert $o$ in $S_{cnd}$
10:      $d = max(|s_{[v,o.x]}|, |s_{[v,o.y]}|)$
11:      if $d < v.key$ then
12:        mark $v$ as dead; break;
13:  if $v$ is not marked dead then
14:    for each unvisited adjacent vertex $v'$ of $v$ do
15:      if $e(v, v')$ does not contain any object then
16:        if $v'$ is not present in the heap $H$ then
17:          insert $v'$ in $H$ with key $v'.key = v.key + |e(v, v')|$
18:        else
19:          $v'.key = min(v'.key, v.key + |e(v, v')|)$
```

that lies on the edge $e(v, v')$ and is closest to v and assign it a safe region $o.s_{[x,y]}$ (lines 7 to 9).

Based on the safe region of the object o, we determine if the vertex v is a dead vertex or not (lines 10 to 12). Recall that a vertex v is a dead vertex if its $SNDist$ from an object o is smaller than its $SNDist$ from the query q. Since, we have assigned safe regions to both the query object q and the data object o, we need to make sure that a vertex v is marked dead only if it satisfies the condition regardless of the location of q and o in their respective safe regions. In other words, a vertex v is marked dead if its maximum

$SNDist$ from the safe region of o is less than its minimum $SNDist$ from the safe region of q. The maximum $SNDist$ of the safe region of o from v is the maximum of the weights of the segments $s_{[v,o.x]}$ and $s_{[v,o.y]}$ where $o.x$ and $o.y$ are the end points of the safe region of o (line 10). To be more precise, this gives an upper bound on the maximum $SNDist$ between v and the safe region of o. The upper bound on the minimum distance of v from the safe region of q is the key value $v.key$ of v (the value with which it was inserted in heap).

We use d to denote the maximum $SNDist$ between v and the safe region of o (line 10). The value of d is compared with the key $v.key$ of the vertex v. If $v.key$ is greater than d, the vertex is marked as dead (lines 11 and 12). Please note that the vertex v will remain dead as long as both the query object and the object o remain in their respective safe regions.

If the vertex v is marked dead, we do not need to consider other adjacent vertices and the objects on the adjacent edges (Lemmas 3.4.1, 3.4.2 and 3.4.3). If the vertex v is not a dead vertex, then each of its adjacent vertex v' that has not been visited earlier is considered (line 14). If the edge $e(v, v')$ contains at least one object, the vertex v' is ignored (line 15). This is because if the shortest path of v' from q passes through v, the vertex v' is a dead vertex because an object o exists on the shortest path. If the edge $e(v, v')$ does not contain any object and v' is not present in the heap then v' is inserted in the heap with key set to $v.key + |e(v, v')|$ (line 17). On the other hand, if v' is already present in the heap then its key is updated to $v.key + |e(v, v')|$ if $v.key + |e(v, v')|$ is less than its existing key (line 19). The algorithm stops when the heap becomes empty.

The edges and segments that are explored during the execution of the algorithm form the unpruned network. Fig. 3.23 shows an example of filtering phase called for the query q. The unpruned network is shown in thick lines. The objects o_1, o_2 and o_3 are the candidate objects.

Proof of correctness can be obtained by proving that the algorithm shortlists every object that may possibly be the RNN of q (by applying Lemma 3.4.1 and Lemma 3.4.2).

We omit the details of the proof. However, it is important to mention that the key $v.key$ of a vertex v may not necessarily be the shortest network distance of v from the safe region of q because the dead vertices are not inserted in the heap. However, this does not affect the correctness of the algorithm because if the shortest path between an object and the query passes through a dead vertex, the object cannot be the RNN. Hence, if $v.key$ is not the shortest network distance between v and the safe region of q then this implies that v is a dead vertex (Lemma 3.4.3) and we do not miss any possible RNN of q.

3.4.5 Verification

An object o is the RNN of q if and only if there does not exist any other object o' such that $SNDist(o, o') < SNDist(o, q)$. If there exists such an object o', the object o is not the RNN and we say that the object o' invalidates the candidate object o.

A straight forward approach to check if a candidate object o is the RNN is to issue a *boolean range query* on the spatial network with range set to $SNDist(o, q)$. A conventional range query returns every object o' for which $SNDist(o, o')$ is less than a given range r. In contrast to conventional range queries, a boolean range query returns true if there exists at least one object o' for which $SNDist(o, o')$ is less than r otherwise it returns false. To check if an object o is the RNN of q, a boolean range query can be issued with range set to $SNDist(o, q)$. If the boolean range query returns true, the object is not the RNN. Otherwise o is reported as the RNN.

Next, we show that some candidate objects may be verified without issuing a boolean range query.

As implied by Lemma 3.4.4, a candidate object o that lies on an edge $e(v_1, v_2)$ that contains at least one other object o' and does not contain q cannot be the RNN of q if $|e(v_1, v_2)| \leq SNdist(v_1, q)$ and $|e(v_1, v_2)| \leq SNDist\ (v_2, q)$. Hence, we compare $|e(v_1, v_2)|$ with the shortest network distances of v_1 and v_2 from the safe region of q and if the edge satisfies the above conditions then the object o does not require verification. For each

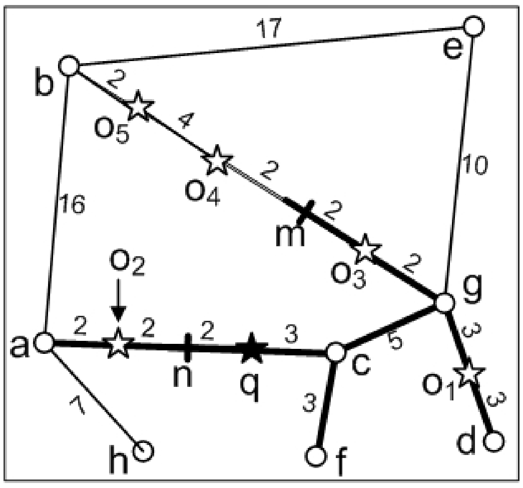


Figure 3.23: Illustration of the filtering phase

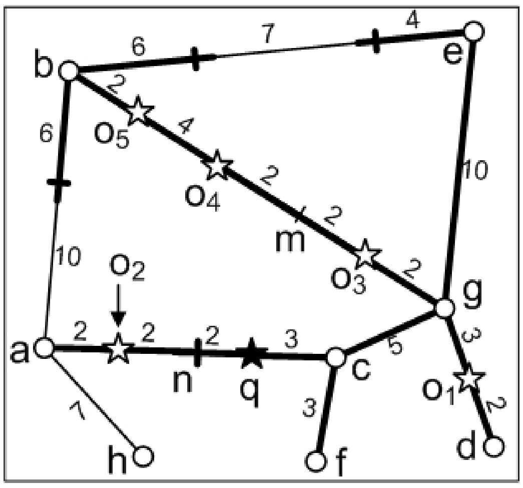


Figure 3.24: Computing monitored network

such candidate object o, we keep a counter that records the number of objects on the edge $e(v_1, v_2)$ and we verify the object o only if the counter is equal to one (i.e., o is the only object on this edge).

There may be several candidates that cannot be verified by using the strategy presented above. One possible way to verify such a candidate object is to issue a boolean range query. Note that the cost of the verification phase may dominate the cost of the filtering phase if a boolean range query is issued for each candidate object. Since verification is to be called at each timestamp regardless of the underlying data movement, the safe region based approach may not improve the performance significantly if the verification phase is expensive.

Next, we present a technique based on the concept of *monitored network*. Once the monitored network for an object o is computed, the verification becomes computationally cheap. We show that the monitored network of a candidate object does not require to be recomputed at every timestamp. In fact, the monitored network of a candidate object remains valid as long as the unpruned network (obtained during the filtering phase) remains valid. In other words, the monitored network is required to be computed only when the filtering phase is called.

Let o be an unverified candidate object. Let $o.s_{[x,y]}$ and $q.s_{[x,y]}$ denote the safe regions of the object o and the query q, respectively. We use $MaxSNDist(o, q)$ to denote the maximum $SNDist$ between the safe regions of o and q (i.e., maximum $SNDist$ between any two points a and b where a is a point in $o.s_{[x,y]}$ and b is a point in $q.s_{[x,y]}$). In the example of Fig. 3.24, the safe regions of o_3 and q are $s_{[m,g]}$ and $s_{[n,c]}$, respectively. The $MaxSNDist(o_3, q)$ is 14 (i.e., the shortest network distance between m and n).

Monitored network of an object o is the part of the network such that for every point p that does not lie on it, minimum $SNDist$ between p and the safe region of o is greater than $MaxSNDist(o, q)$.

Fig. 3.24 shows the monitored network of object o_3 (shown in thick lines) and it consists of every point of the network that has minimum $SNDist$ from the safe region of o_3 at most 14. Intuitively, the monitored network is defined as the network such that, for any object o' that does not lie on it, $SNDist(o, o') > SNDist(o, q)$ regardless of the locations of o and q in their respective safe regions. Hence, only the objects that lie on the monitored network are required to be considered in order to verify the candidate object o.

Note that once the monitored network is computed it remains valid as long as q and o remain in their respective safe regions. Hence, the recomputation of the monitored network is not required unless at least one of q and o does not leave its safe region. Recall that if q or o leaves its safe region, the filtering phase is required to be called again. Hence, the monitored network remains valid as long as the filtering phase is not required to be called again.

To compute the monitored network, we use an algorithm similar to Dijkstra's algorithm and Algorithm 4 and expand the network starting from the safe region of o until we visit every vertex v such that minimum $SNDist$ of v from the safe region of o is at most equal to $MaxSNDist(o, q)$.

To enable efficient computations of $SNDist$ between o and other objects in the monitored network, we maintain the minimum $SNDist$ of each explored vertex from the safe region of o. In addition, we also maintain the list of objects that lie on the monitored network of o. An object o' notifies the server when it enters or leaves the monitored network of the object o and the list of the objects that lie on the monitored network of o is updated accordingly.

Optimizations. We present two optimizations that can help in terminating the computation of the monitored network earlier.

1. During the computation of the monitored network, we do not need to expand the network beyond a vertex v if the shortest path between v and the safe region of o contains the query q. This is because any object o' that lies beyond the vertex v satisfies $SNDist(o, o') > SNDist(o, q)$. For example, in Fig. 3.24, an object that lies on segment $s_{[q,a]}$ (e.g., o_2) cannot help in verifying the object o_3 because the shortest path from the safe region of o_3 to the segment $s_{[q,a]}$ contains the query q. Hence, we do not need to include this segment in the monitored network.

2. Similar to the definition of $MaxSNDist(o, q)$, we define $MinSNDist(o, q)$ as the minimum $SNDist(o, q)$ between the safe regions of o and q. In the example of Fig. 3.24, $MinSNDist(o_3, q)$ is 5 and $MaxSNDist(o, q)$ is 14 where the safe regions of o_3 and q are $s_{[m,g]}$ and $s_{[n,c]}$, respectively.

Note that if there exists an object o' such that it satisfies $MaxSNDist(o, o') < MinSNDist(o, q)$ then the object o cannot be the RNN of q as long as the objects o and o' and the query q remain in their respective safe regions. We utilize this observation and expand the monitored network such that it covers every point p such that $SNDist(o, p) < MaxSNDist(o, o')$. Here o' corresponds to the object that has smallest $MaxSNDist(o, o')$ among all the objects discovered so far. For instance, in Fig. 3.24, $MaxSNDist(o_3, o_4)$ is 12 (the safe region of o_4 is $s_{[b,m]}$) and $MaxSNDist(o_3, o_1)$ is 9. Hence, during the computation of the monitored network, we may stop expanding the network when the expanded network contains every point p that has minimum $SNDist$ from the safe region of o_3 at most 9. To verify the object o, we compute $SNDist$ between o and every other object (including the query) that lies on the expanded network.

Note that if we use the optimization presented above, the computed monitored network does not need to be updated as long as o, o' and q remain in their respective safe regions. If o' is also a candidate object (e.g., $o' = o_3$ in Fig. 3.24), the monitored network remains valid as long as the filtering phase is not called again. Otherwise, when the object o' moves out of its safe region the monitored network is recomputed to guarantee the correctness.

3.4.6 Safe regions consisting of more than one edges

In this section, we show that our proposed algorithm can support the safe regions consisting of more than one edges and segments. Assume that the safe region of an object o consists of more than one edges and segments. We denote its safe region $o.s$ by its end points (i.e., boundary points). Consider the example of Fig. 3.23 and assume that the safe region of the object o_3 is shown in thick lines. The end points of the safe region of o_3 are m, g, d, f and a.

The safe region of a query q is always chosen such that it does not overlap with the safe region of any other object. The algorithm 4 is modified as follows. At line 3, all the end points of the safe region of q are inserted in the heap with keys set to zero. Moreover, the vertices that lie inside the safe region of q are marked as visited so that they are not considered during the network expansion. Note that as the key of every end point of $q.s$ is set to zero, the key of a de-heaped vertex v denotes minimum $SNDist$ from the safe region of q to the vertex v. At line 10, d is set as the maximum distance between v and the safe region of o. Note that these changes guarantee that a vertex v remains dead as long as q and o remain inside their respective safe regions.

The rest of the filtering algorithm does not require any changes. The techniques and optimizations we presented in the verification phase can be immediately applied and do not require any change.

3.4.7 Extensions

Queries on directed graphs

In the previous section, our main focus was on the RNN queries in the spatial networks that are represented by undirected graphs. Our proposed techniques can be easily extended for the directed graphs. Below, we highlight the changes we need to make to extend our techniques for the RNN queries on directed graphs.

1. $SNDist(x, y)$ is defined as the total weights of the edges and the segments on the shortest path *from* the point x *to* the point y.

2. Lemma 3.4.1 and Lemma 3.4.2 are the same except that we use the shortest path *from* o *to* q instead of the shortest path between q and o. The definition of the dead vertex remains unchanged.

3. Lemma 3.4.3 is the same except that we use shortest path *from* v' *to* q instead of shortest path between q and v'.

4. Lemma 3.4.4 is not applicable whereas Lemma 3.4.5 and Lemma 3.4.6 remain unchanged.

5. Filtering phase (Algorithm 4) is similar except that we expand the network from any vertex v to v' (see line 6) only if there is a directed edge from the vertex v' to v. Please note that we expand the network in the direction opposite to the direction of edges. This is because the lemmas are applicable on the path from an object o to q and not on the path from q to o.

6. The verification phase remains same and we compute the monitored network by expanding the network in the direction of the edges.

RkNN queries

We briefly discuss the necessary changes that are required to extend our proposed techniques for RkNN queries.

1. Lemma 3.4.1 is restated as follows. An object o cannot be the RkNN of q if the shortest path from o to q contains *at least* k other objects.

2. A dead vertex v is redefined as a vertex v for which there exist at least k objects such that for every such object o, $SNDist(v, o) < SNDist(v, q)$. After redefining the dead vertices, Lemma 3.4.2 and Lemma 3.4.3 do not require any modification.

3. Lemma 3.4.4 is the same except that it is applicable only on the edges that contain *at least* $(k + 1)$ objects on it. Recall that Lemma 3.4.4 is only applicable for RkNN queries on undirected graphs. For the RkNN queries on the directed graphs, we do not consider Lemma 3.4.4.

4. Extreme objects are redefined. An object o is called an extreme object of an edge $e(v_1, v_2)$ if either the segment $s_{[o,v_1]}$ or the segment $s_{[o,v_2]}$ contains *at most* $k - 1$ other objects. Lemma 3.4.5 holds after the extreme objects are redefined as above.

5. Lemma 3.4.6 is restated as follows. Regardless of the number of queries in the system. An edge that does not contain any query has at most $2k$ objects that can be the RkNNs of any of the queries.

6. Filtering phase is similar except that we mark the vertices as dead according to the redefined definition of the dead vertices.

7. Verification phase is similar except that an object o is reported as RkNN iff there are at most $k - 1$ other objects closer to o than q. Computation of the monitored network remains unchanged.

Bichromatic queries

Let P and O be the two sets of objects and assume that the query q belongs to O. In the filtering phase, only the objects of type O are considered to prune the network. The objects of type O that are discovered during the filtering phase are called *filtering* objects. The objects of type P that lie on the unpruned network are called the candidate objects. The set of candidate objects remain valid unless at least one of the following three happens: i) the query leaves its safe region; ii) one of the filtering objects leaves its safe region; iii) one of type P objects enters or leaves the unpruned network. In the case when one of the first two events happens, the filtering and verification phases are called. If only the third event happens, we do not need to call the filtering phase again because the unpruned network is not affected by the movement of type P objects. Instead, we update the set of candidate objects by adding the objects of type P that enter the unpruned network and removing the type P objects that leave the unpruned network.

3.5 Experiment Results

All the experiments were conducted on Intel Xeon 2.4 GHz dual CPU with 4 GBytes memory. All the algorithms (including the competitors) were implemented in C++. Our algorithm is called SAC (**S**wift **A**nd **C**heap) due to its computational efficiency and communication cost saving.

As discussed in Section 3.2.3, there may be some applications where the objects have to report their locations to the server for other types of queries like range queries, nearest neighbor queries etc. In such case, the server is responsible for checking whether an object lies in the safe region or not. In order to show the superiority of our technique in all kinds of applications, the computation costs shown in the experiments include the cost of checking whether each object lies in its safe region or not. Obviously, the computation cost would be less for the case when the clients report their locations only when they leave their safe regions.

In Section 3.5.1, we evaluate the performance of our Euclidean space algorithm. The performance of our spatial network algorithm is evaluated in Section 3.5.2.

3.5.1 Query Processing in Euclidean Space

For RNN queries ($k = 1$), we compare our algorithm with state-of-the-art algorithm (IGERN) [KMS+07] which has been shown superior in [KMS+07] to other RNN monitoring algorithms [XZ06, WYCT08a]. For RkNN queries ($k > 1$), we compare our algorithm with CRkNN [WYCT08a] which is the only available RkNN monitoring algorithm. In accordance with work in [KMS+07] and [WYCT08a], we choose 64 × 64 grid structure for IGERN and 100 × 100 grid structure for CRkNN. For our algorithm, the grid cardinality is 64 × 64.

Similar to previous work, we simulated moving cars by using the spatio-temporal data generator [Bri02]. Input to the generator is road map of Texas[4] and output is a set of cars (objects and queries) moving on the roads. The size of data universe is 1000 Km × 1000 Km. The parameters of data sets are shown in Table 3.2 and default values are shown in bold.

Parameter	**Range**
Number of objects (×1000)	40, 60, 80, **100**, 120
Number of queries	100, 300, **500**, 700, 1000
Average speed (in Km/hr)	40, 60, **80**, 100, 120
Side length of safe region (in Km)	0.2, 0.5, **1**, 2, 3, 4
Mobility (%)	5, 20, 40, 60, **80**, 100

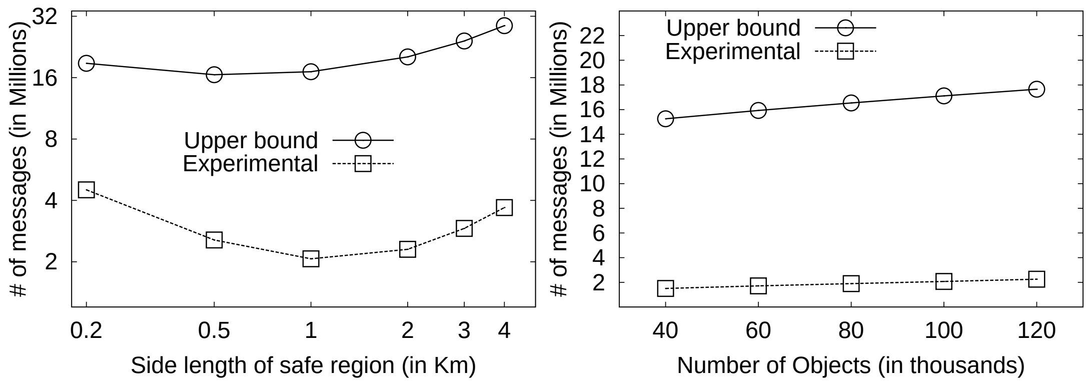
Table 3.2: System parameters for experiments in Euclidean space

The server reports the results continuously after every one second (i.e., the timestamp length is 1 sec). Both the objects and queries are cars moving on roads and they have similar properties (e.g., average speed, mobility). Mobility refers to the percentage of objects and queries that are moving at any timestamp (percentage of objects and queries that change their locations between two consecutive timestamps). All queries are continuously monitored for five minutes (300 timestamps) and the results shown correspond to total CPU time and communication cost. Communication cost is the total number of messages sent between clients and server.

In Fig. 3.25, we conduct experiments to verify the cost analysis presented in Section 3.3.4. The experiments show that the actual cost is around 12% to 25% of the upper bound. We also observe that the actual results follow a trend similar to the trend anticipated by our theoretical analysis (e.g., the cost increases if the safe region is too small or too large).

Fig. 3.26(a) shows the effect of the safe region size on computation time our algorithm

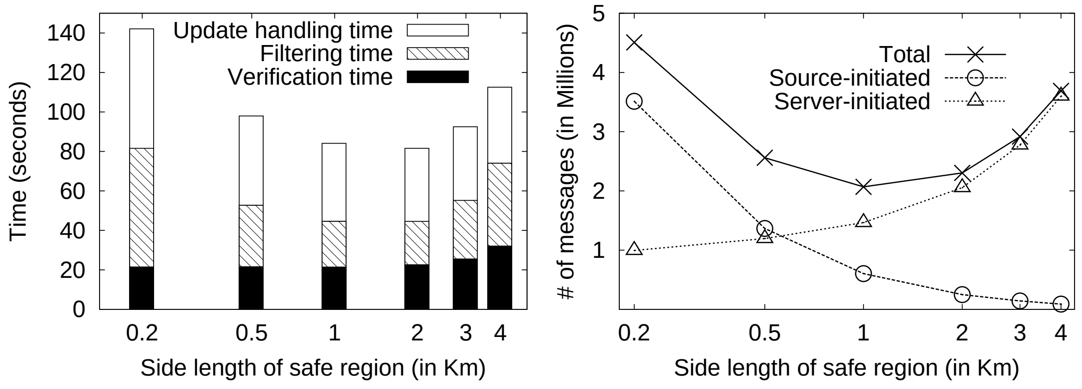
[4]http://www.census.gov/geo/www/tiger/



(a) Varying safe region (b) Varying data size

Figure 3.25: Verifying theoretical upper bound

and IGERN [KMS+07]. The computation cost consists of update handling cost, filtering cost and verification cost. The update handling cost includes the cost of checking whether an object/query is in its safe region or not and updating the underlying grid structure if the object/query leaves the safe region. If the safe region is too small, the set of candidate objects is affected frequently and the filtering is required more often. Hence, the cost of the filtering phase increases. On the other hand, if the safe region is too large, the number of candidate objects increases and the verification of these candidates consumes more computation time. Also, the cost of filtering phase increases because less space can be pruned if the safe region is large. The update handling cost is larger for smaller safe regions because the objects and queries leave the safe regions more frequently.



(a) Computation time (b) Communication cost

Figure 3.26: Effect of safe region size

Fig. 3.26(b) studies the effect of safe region size on communication cost. As studied in Section 3.3.4, the number of source-initiated updates increases if the side length of the safe region is small. On the other hand, if the safe region is large, the number of server-initiated updates increases. Fig. 3.26(b) verifies this. In current experiment settings, our algorithm performs best when the side length of the safe region is 1Km so we choose this value for the remaining experiments.

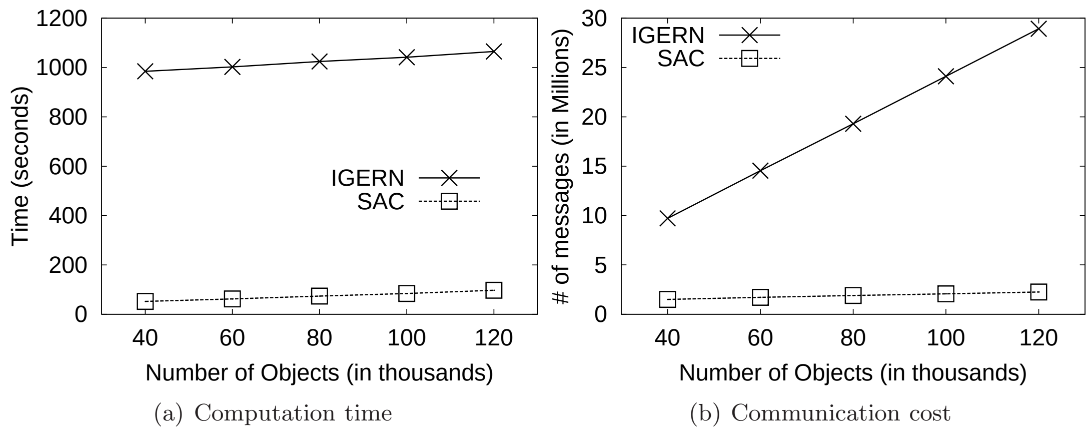
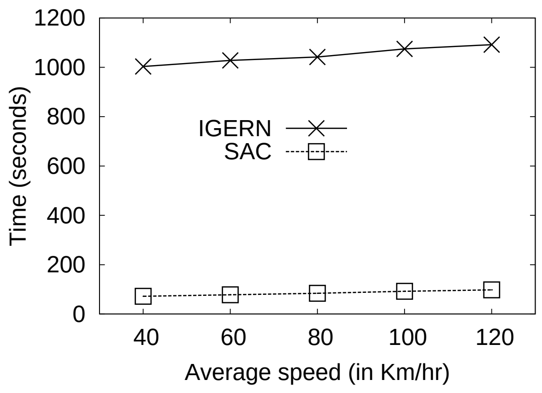


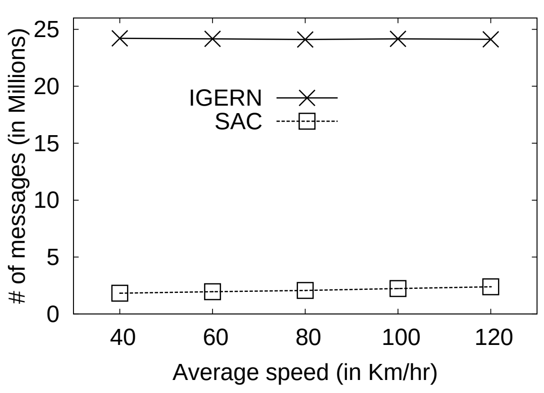
Figure 3.27: Effect of data size

Fig. 3.27 shows the effect of the number of objects. Our algorithm not only outperforms IGERN but also scales better. The composition of CPU time is not shown due to the huge difference in the performance of both algorithms. However, the composition of CPU time is similar to Fig. 3.26(a) for our algorithm. For IGERN, the filtering phase takes 95% to 99% of the total cost in all experiments. This is because the expensive filtering phase is called frequently.

Fig. 3.28 studies the effect of the average speed of queries and objects. Fig. 3.28(a) shows that the computation time increases for both of the approaches as the speed increases. For our approach, the time increases because the objects and queries leave their respective safe regions more frequently and the filtering phase is called more often. Fig. 3.28(b) shows that IGERN requires an order of magnitude more messages than our approach. The communication cost for our approach increases due to the larger number of source-initiated updates as the speed increases.

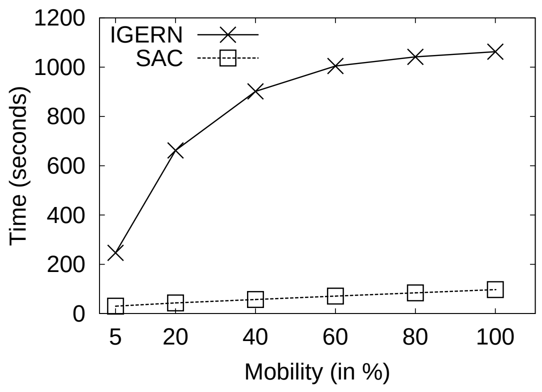


(a) Computation time

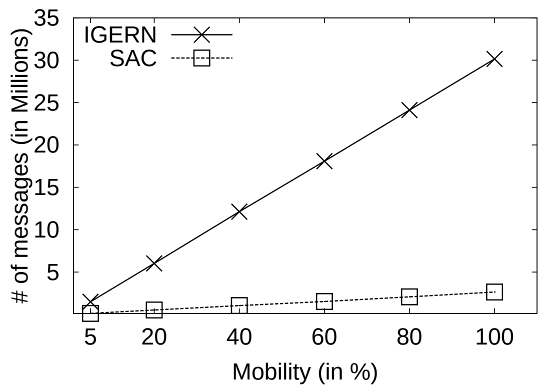


(b) Communication cost

Figure 3.28: Effect of Speed



(a) Computation time

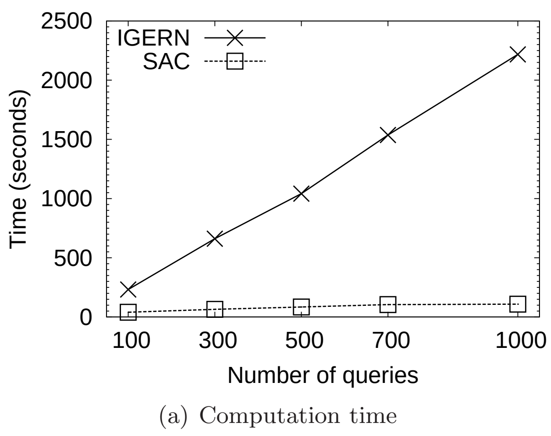


(b) Communication cost

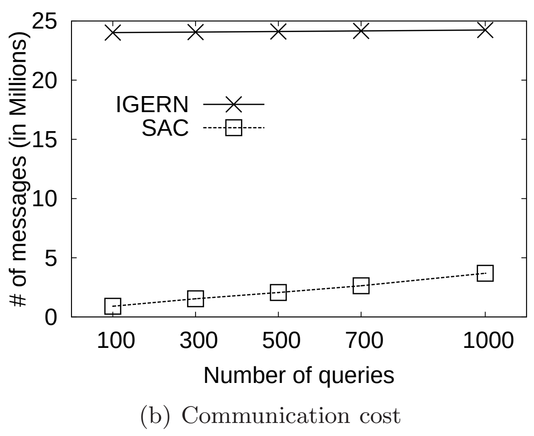
Figure 3.29: Effect of data mobility

Fig. 3.29(a) compares the computation time for increasing data mobility. As expected, IGERN performs good when the object mobility is low (e.g., 5%). However, its computation cost increases significantly as the object mobility increases. Our algorithm performs better for all cases and scales decently. Fig. 3.29(b) studies the effect of objects and queries mobility on the communication cost. Since only the moving objects report their locations, the number of messages increase with the increase in mobility. However, our algorithm consistently gives improvement of more than an order of magnitude compared to IGERN.

Fig. 3.30 studies the effect of number of queries. In Fig. 3.30(a), we note that our algorithm gives more than an order of magnitude improvement over IGERN in terms of



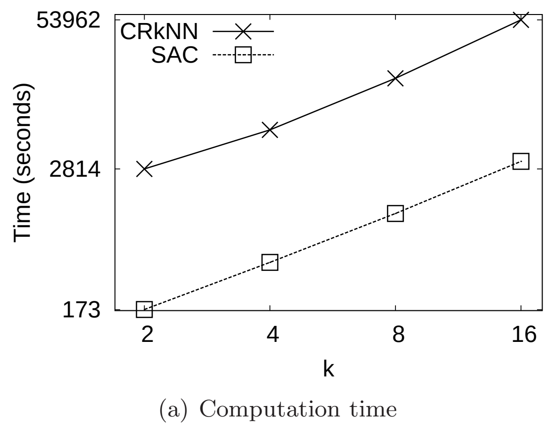
(a) Computation time



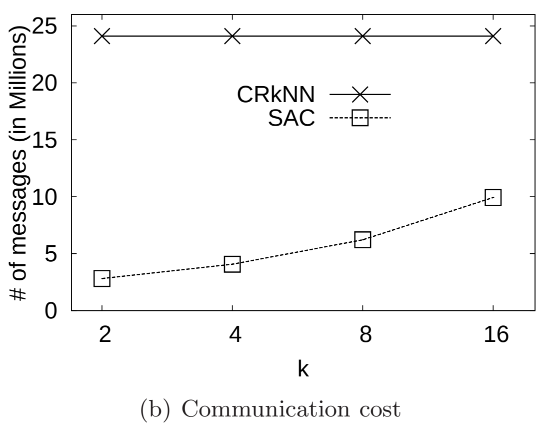
(b) Communication cost

Figure 3.30: Effect of number of queries

CPU time and scales better. In accordance with the analysis in Section 3.3.4, Fig. 3.30(b) show that the communication cost of our approach increases with the number of queries.



(a) Computation time



(b) Communication cost

Figure 3.31: Effect of k

Fig. 3.31 studies the effect of k on communication and computation time. Fig. 3.31(a) compares our approach with [WYCT08a] referred as CRkNN. Computation cost of both approaches increases with increase in k. However, our algorithm scales better (note the log scale). CRkNN continuously monitors $6k$ range queries to verify the candidate objects. To monitor these queries, it keeps a counter for the number of objects leaving and entering within the range. However, this information becomes useless when the candidate object or query changes its location. As shown in Fig. 3.31(b), communication cost for our approach increases for larger values of k. This is mainly because the number of candidate objects that require verification increases with k. Communication cost of our algorithm reaches to 24 million when $k = 64$ (CPU time $23,000$ sec). We were unable to run CRkNN for $k > 16$ due to its large main-memory requirement.

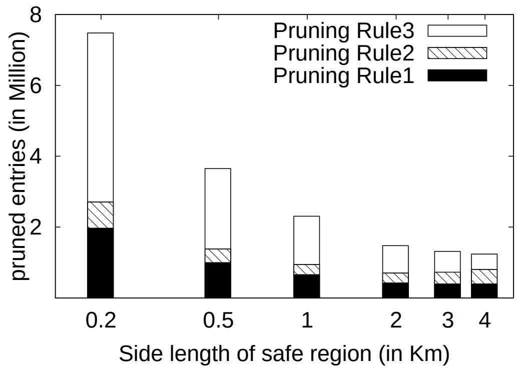


Figure 3.32: Effectiveness of pruning rules

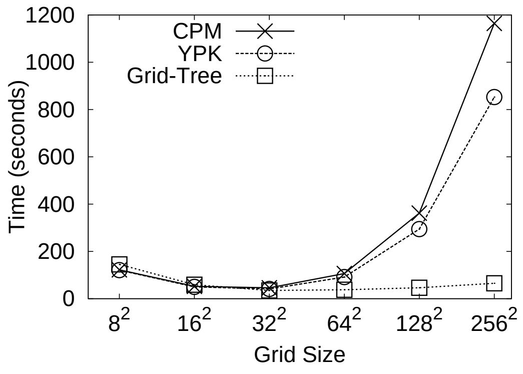


Figure 3.33: Effectiveness of grid-tree

Fig. 3.32 shows the effectiveness of pruning rules for different safe region sizes. Pruning rules are applied in the same order as in Algorithm 1. If a pruning rule fails to prune an entry (an object or a node of the grid-tree), the next pruning rule is used to prune it. Fig. 3.32 shows that a greater number of entries are pruned if the safe region size is small. Majority of the entries are pruned by the metric based pruning (pruning rule 3) when the safe regions are small. The average time to prune an entry by metric based pruning, dominance pruning and half space pruning is 1.1, 2.3 and 10.5 micro seconds, respectively.

Now, we show the effectiveness of grid-tree over previous proposed grid access methods CPM [MHP05] and YPK [YPK05]. Fig. 3.33 shows the total CPU time for our RNN monitoring algorithm when the underlying constrained nearest neighbor algorithm (and marking and unmarking of cells) use CPM, YPK and grid-tree. We change the grid size from 8×8 to 256×256. Grid-tree based RNN monitoring algorithm scales much better with increase in number of cells.

3.5.2 Query Processing in Spatial Networks

To the best of our knowledge, we are the first to propose an algorithm to continuously monitor RNN queries in spatial networks for the case where both the queries and data objects continuously change their locations. We compare our algorithm (SAC) with a naïve algorithm. The safe regions used by SAC consist of at most one edge. Naïve algorithm recomputes the results at every timestamp by applying our algorithm and setting the safe region size to zero (i.e., safe region is not used). We choose a better competitor of our algorithm and call it NSR (No Safe Region). NSR is the same as naïve algorithm except that it calls the filtering phase only when the query object or one of the candidate objects changes its location. As obvious, the naïve algorithm performs worse than NSR. Hence, we compare our algorithm with NSR.

We use the road network of California[5] that consists of around $22,380$ road segments (edges). Each object in the data set randomly picks a vertex and starts moving towards it with a certain speed (a system parameter). When the object reaches at its destination vertex, it randomly chooses one of its adjacent vertices and continues travelling towards it. The queries are generated similarly. Table 3.3 shows the default parameters used in our experiments.

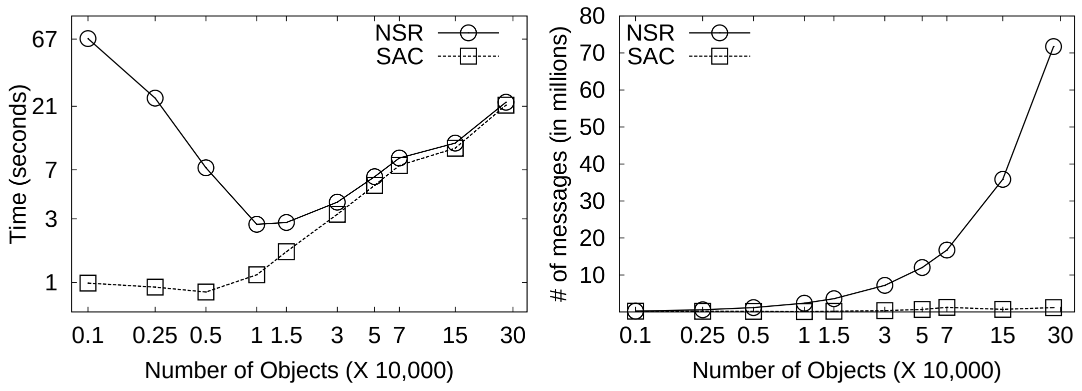
Parameter	**Range**
Number of objects ($\times$1000)	1, 2.5, **5**, 10, 15, 30, 50, 70, 150, 300
Number of queries	25, **100**, 150, 250, 500, 700, 1000
Average speed (in Km/hr)	40, 60, **80**, 100, 120
Mobility (%)	5, 20, 40, 60, **80**, 100

Table 3.3: System parameters for experiments in road network

Each query is continuously monitored for 300 timestamps (five minutes) and the results shown correspond to the total CPU time and total communication cost. The communication cost corresponds to the number of messages sent between the server and clients.

In Fig. 3.34, we study the effect of number of objects on the road network. Fig. 3.34(a)

[5]http://www.cs.fsu.edu/ lifeifei/Spatialdataset.htm



(a) Computation time (b) Communication cost

Figure 3.34: Effect of data size

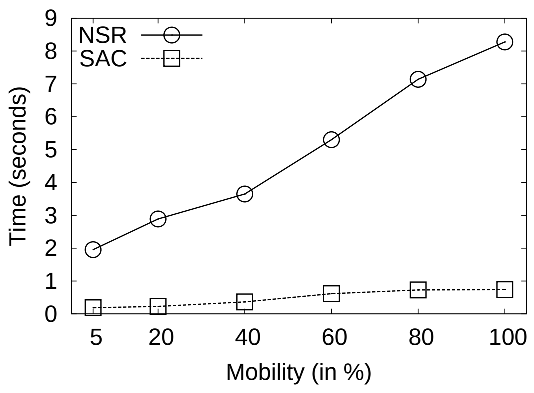
shows the effect on the computation time of both the algorithms (note that log-scale is used). Interestingly, the performance of both the algorithms is poor when the number of objects is too small or too large. When the number of objects is large, the performance becomes poor mainly because updates of more objects are needed to be handled. Since the dominant cost is handling these location updates, both of the algorithms perform similar when the number of objects is large.

When the number of objects is small, greater number of edges are to be explored for filtering and verification phases which results in greater computation time. Note that if each edge contains several objects, the RNN queries can be answered by visiting at most one or two edges. Hence, it would be more interesting to compare the performance of the algorithms where the density of objects (number of objects per edge) is low. For this reason, we choose 5000 objects for the rest of the experiments.

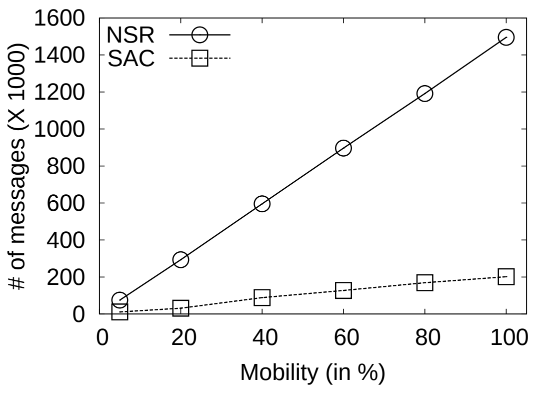
Fig. 3.34(b) shows the trend that the communication costs of both algorithms increase with the increase in number of objects (log scale is used for x-axis). However, the safe region based algorithm SAC scales much better than NSR.

Fig. 3.35 studies the effect of data mobility on both of the algorithms. As expected, both algorithms perform worse as the data mobility increases. However, SAC scales much better than NSR both in terms of computation time and communication cost.

Fig. 3.36 studies the effect of speed of the objects and queries on the algorithms.

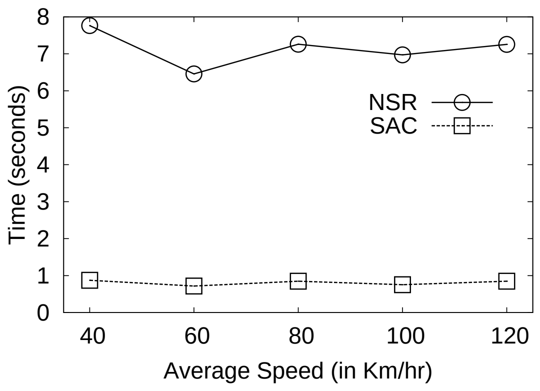


(a) Computation time

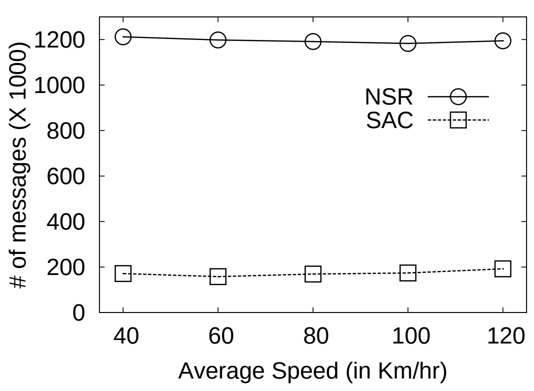


(b) Communication cost

Figure 3.35: Effect of data mobility



(a) Computation time

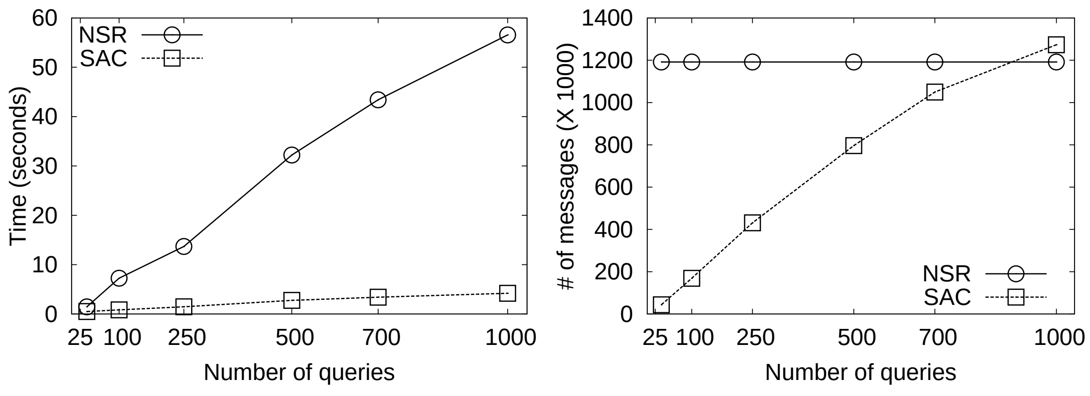


(b) Communication cost

Figure 3.36: Effect of speed

The experiments demonstrate that the performance of the proposed technique is not significantly affected by the speed. Although the objects and queries leave their safe regions more frequently as the speed increases, the communication cost is not significantly affected. This is because the total communication cost is dominated by the server initiated updates (e.g., when the server requests the objects to send their exact locations in order to verify if a candidate is the RNN or not). The number of server initiated updates does not depend on the speed hence the total communication cost is not significantly affected by the speed.

In Fig. 3.37, we change the number of queries and show the effect on the performances of both the algorithms. The computation times for both of the algorithms increase



(a) Computation time (b) Communication cost

Figure 3.37: Effect of number of queries

as the number of queries increases but SAC scales much better. The communication cost of NSR does not depend on the number of queries because each object reports its location whenever it changes its location. On the other hand, the communication cost of SAC increases mainly because more objects are required to be verified if the number of queries is large. To verify more objects, greater number of server initiated updates are required and this results in increased communication cost. Fig. 3.37(b) shows that the communication cost of SAC is more than the cost of NSR when a large proportion of the data objects are also the query objects (e.g., 1000 queries among 5000 objects).

We remark that in the worst case the communication cost of SAC can be at most two times the cost of NSR. This is because, for each object, at most two messages are to be sent (one to request the location and one to receive the server). Nevertheless, in the applications where the proportion of queries is large, the clients (data objects) may be configured to send their locations at every timestamp. The communication cost in this case would be the same as the cost of NSR.

Fig. 3.38 shows the average time taken by a call to filtering phase, a call to compute the monitored network (shown as MN computation) and a call to verify the objects (after the monitored network has been computed). As expected, the cost of filtering and computing the monitored network is high if the number of objects is too large or too small. The cost of verification increases when the data size is too small. This is mainly

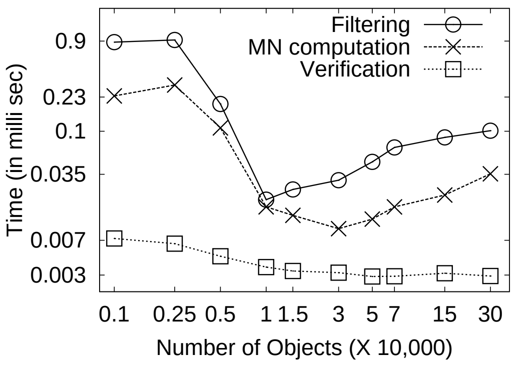


Figure 3.38: Costs of different phases

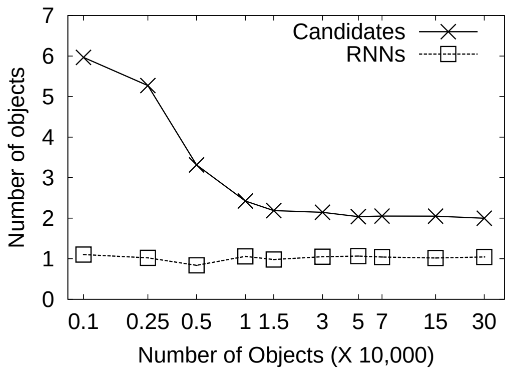


Figure 3.39: Average number of candidates

because the number of candidate objects increases when the data size is small. The next experiment confirms this trend.

In Fig. 3.39, we study the effect of data size on the number of candidate objects and the number of RNNs. We observe that the number of candidate objects is large when the data size is small. This is because the algorithm needs to explore more edges during the filtering phase and the pruning power decreases.

3.6 Summary

In this chapter, we studied the problem of continuous monitoring of reverse k nearest neighbors queries in Euclidean space as well as in spatial networks. Existing techniques are sensitive towards objects and queries movement. For example, the results of a query are to be re-computed whenever the query changes its location. We present a framework for continuous reverse k nearest neighbor (RkNN) queries by assigning each object and query with a safe region such that the expensive recomputation is not required as long as the query and objects remain in their respective safe regions. This significantly improves the computation cost. As a by-product, our framework also reduces the communication cost in client-server architectures because an object does not report its location to the server unless it leaves its safe region or the server sends a location update request. We also conduct a rigid cost analysis for our Euclidean space RkNN algorithm. We show that our techniques can also be applied to answer bichromatic RkNN queries in Euclidean space as well as in spatial networks. Furthermore, we show that our techniques can be extended for the spatial networks that are represented by directed graphs. The extensive experiments demonstrate that our techniques outperform the existing techniques by an order of magnitude in terms of computation cost and communication cost.

Chapter 4

Influence Zone Based Processing of RkNN Queries

Chapter 3 presented our technique (Lazy Updates) to answer continuous RkNN queries in Euclidean space and in spatial networks. Lazy Updates can be used to answer RkNN queries for the case when all the queries and the data objects are moving. In many applications, the queries do not change their locations. In this chapter, based on a novel concept of *influence zone*, we present a more efficient technique for the case when only the data objects move whereas the queries do not change their locations. Our influence zone based approach can also be used to efficiently answer snapshot RkNN queries and perform all existing techniques. This chapter is based on our research reported in [CLZZ11, CLZZ].

4.1 Overview

As stated earlier, a RkNN query finds every data point for which the query point q is one of its k nearest neighbors. Since q is close to such data points, q is said to have high influence on these points. Hence, the set of points that are the RkNNs of a query is called its influence set [KM00]. Consider the example of a gas station. The drivers for which this gas station is one of the k nearest gas stations are its potential customers.

Throughout this chapter, the objects that provide a facility or service (e.g., gas stations) are called *facilities* and the objects (e.g., the drivers) that use the facility are called *users*. The influence set of a given facility q is then the set consisting of every user for which q is one of its k closest facilities.

In this chapter, we first introduce a more generic concept called *influence zone* and then we show that the influence zone can be used to efficiently compute the influence set (i.e., RkNNs). Consider a set of facilities $F = \{f_1, f_2, \cdots, f_n\}$ where f_i represents a point in Euclidean space and denotes the location of the i^{th} facility. Given a query $q \in F$, the influence zone Z_k is the area such that for every point $p \in Z_k$, q is one of its k closest facilities and for every point $p' \notin Z_k$, q is not one of its k closest facilities.

The influence zone has various applications in location based services, marketing and decision support systems. Consider the example of a coffee shop. Its influence zone may be used for market analysis as well as targeted marketing. For instance, the demographics of its influence zone may be used by the market researchers to analyse its business. The influence zone can also be used for marketing, e.g., advertising bill boards or posters may be placed in its influence zone because the people in this area are more likely to be influenced by the marketing. Similarly, the people in its influence zone may be sent SMS advertisements.

Note that the concept of the influence zone is more generic than the influence set, i.e., the RkNNs of q can be computed by finding the set of users that are located in its influence zone. We show that our influence zone based RkNN algorithms significantly outperform existing best known algorithms for both the *snapshot* and *continuous* RkNN queries.

Existing RkNN processing techniques [SAA00, TPL04, WYCT08b, KMS+07] (including Lazy Updates presented in Chapter 3) require a *verification* phase to answer the queries. Initially, the space is pruned by using the locations of the facility points. Then, the users that are located in the unpruned space are retrieved. These users are the possible RkNNs and are called candidates. Finally, in the verification phase, a range query is issued for every candidate to check if it is a RkNN or not.

In contrast to the existing approaches, our influence zone based algorithm does not require the verification phase. Initially, we use our algorithm to efficiently compute the influence zone. Then, every user that is located in the influence zone is reported as RkNN. This is because a user can be the RkNN if and only if it is located in the influence zone. Similarly, to continuously monitor RkNNs, initially the influence zone is computed. Then, to update the results, we only need to monitor the users that enter or leave the influence zone (i.e., the users that enter in the influence zone become the RkNNs and the users that leave the influence zone are no more the RkNNs). To further improve the performance, we present efficient methods to check whether a point lies in the influence zone or not.

It is important to note that the influence zone of a query is the same as the Voronoi cell of the query when $k = 1$ [SRAA01]. For arbitrary value of k, there does not exist an equivalent representation in literature (i.e., order k Voronoi cell is different from the influence zone). Nevertheless, we show that a precomputed order k Voronoi diagram can be used to compute the influence zone (see Section 4.5.1). However, using the precomputed Voronoi diagrams is not a good approach to process spatial queries as mentioned in [ZZP+03]. For instance, the value of k is not known in advance and precomputing several Voronoi diagrams for different values of k is expensive and incurs high space requirement. In Section 4.5.1, we state several other limitations of this approach.

Below, we summarize our contributions.

- We present an efficient algorithm to compute the influence zone. Based on the influence zone computation algorithm, we present efficient algorithms that outperform best known techniques for both *snapshot* and *continuous* RkNN queries.

- We provide a detailed theoretical analysis to analyse the IO costs of our influence zone and RkNN computation algorithms, the area of the influence zone and the number of RkNNs. Our experiment results show the accuracy of our theoretical analysis.

- Our main algorithm uses an algorithm similar to the one proposed in [WYCT08b]. It was shown that the complexity of that algorithm is $O(m^2)$ [WYCT08b] where m is the number of facilities used to prune the search space. We conduct a rigorous complexity analysis and show that the complexity of the algorithm can be reduced to $O(km)$ when k is smaller than m.

- We demonstrate that the influence zone computation technique can be extended for dimensionality higher than two. We also present techniques to efficiently update the influence zone when the underlying data set is updated by insertions or deletions.

- Our extensive experiments on real and synthetic data demonstrate that our proposed algorithms are several times faster than the existing best known algorithms for both the snapshot and continuous RkNN queries.

This chapter is organized as follows. In Section 4.2, we formally define the problem using specific terms and notations that ease the presentation of our techniques. Section 4.3 presents our technique to efficiently compute the influence zone. In Section 4.4, we present efficient techniques to answer RkNN queries by using the influence zone. A detailed theoretical analysis is presented in Section 4.5. The techniques to handle data updates are presented in Section 4.6 followed by the experiment results in Section 4.7. Section 4.8 summarizes this chapter.

4.2 Problem Definition

For the ease of presenting the techniques in this chapter, we redefine the problem of RkNN queries by using different terms and notations. We remark that the problem remains the same as was defined in previous chapters and only the terms and notations used to define it are changed.

Consider a set of facilities $F = \{f_1, f_2, \cdots f_n\}$ and a query $q \in F$ in a Euclidean space[1]. Given a point p, C_p denotes a circle centered at p with radius equal to $dist(p, q)$ where $dist(p, q)$ is the distance between p and q. $|C_p|$ denotes the number of facilities that lie within the circle C_p (i.e., the count of facilities such that for each facility f, $dist(p, f) < dist(p, q)$). Please note that the query q can be one of the k closest facilities of a point p iff $|C_p| < k$. Now, we define influence zone and RkNN queries.

Influence zone Z_k. Given a set of facilities F and a query $q \in F$, the influence zone Z_k is the area such that for every point $p \in Z_k$, $|C_p| < k$ and for every point $p' \notin Z_k$, $|C_{p'}| \geq k$.

Now, we define the reverse k nearest neighbor (RkNN) queries. RkNN queries are classified [KM00] into *bichromatic* and *monochromatic* RkNN queries. Below, we define both.

Bichromatic RkNN queries. Given a set of facilities F, a set of users U and a query $q \in F$, a bichromatic RkNN query is to retrieve every user $u \in U$ for which $|C_u| < k$.

Consider that the supermarkets and the houses in a city correspond to the set of facilities and users, respectively. A bichromatic RkNN query may be used to find every house for which a given supermarket is one of the k closest supermarkets.

Monochromatic RkNN queries. Given a set of facilities F and a query $q \in F$, a monochromatic RkNN query is to retrieve every facility $f \in F$ for which $|C_f| < k + 1$.

Please note that for every f, C_f contains the facility f. Hence we have condition $|C_f| < k + 1$ instead of $|C_f| < k$. Consider a set of police stations. For a given police station q, its monochromatic RkNNs are the police stations for which q is one of the k nearest police stations. Such police stations may seek assistance (e.g., extra policemen) from q in case of an emergency event.

Bichromatic continuous RkNN queries. As mentioned in Chapter 1, in a continuous query, the results are to be continuously updated as the objects in the underlying data sets change their locations. In this chapter, we focus on a special case of continuous RkNN queries where only the users change their locations.

[1] Although, like existing techniques [WYCT08b, CLZ+09], our focus in this chapter is two dimensional location data, in Section 4.3.4, we show that the techniques can be extended to higher dimensionality.

Given a set of facilities F, a query $q \in F$ and a set of users U, a continuous RkNN query is to continuously update the bichromatic RkNNs of q when one or more users change their locations.

A gas station may want to continuously monitor the vehicles for which it is one of the k closest gas stations. It may issue a continuous RkNN query to do so.

Table 4.1 defines other notations used throughout this chapter.

Table 4.1: Notations

Notation	Definition
q	the query point
C_p	a circle centered at p with radius $dist(p, q)$
$\|C_p\|$	the number of facilities located inside C_p
$B_{x:q}$	a perpendicular bisector between point x and q
$H_{x:q}$	a half-plane defined by $B_{x:q}$ containing point x
$H_{q:x}$	a half-plane defined by $B_{x:q}$ containing point q

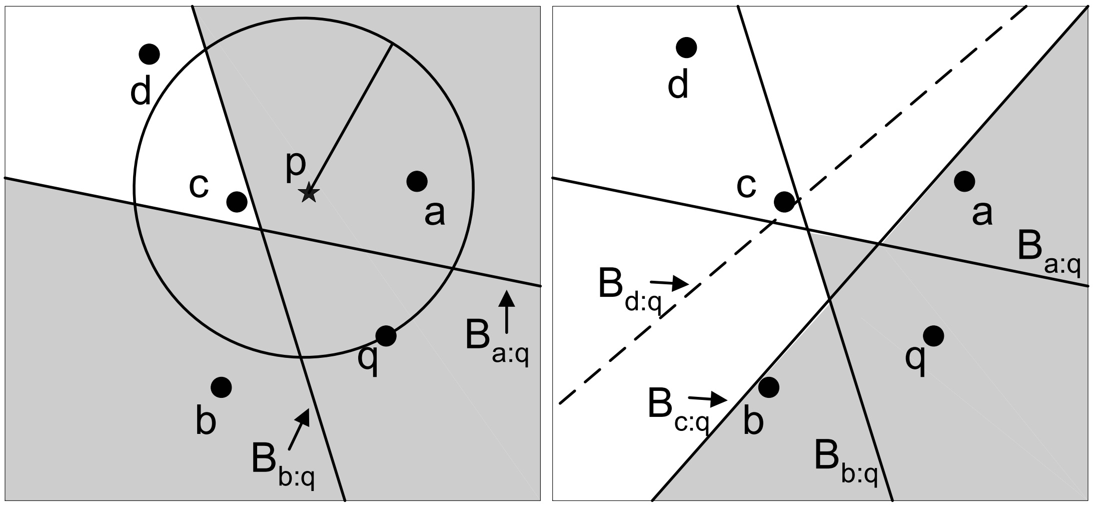
4.3 Computing Influence Zone

4.3.1 Problem Characteristics

Given two facility points a and q, a perpendicular bisector $B_{a:q}$ between these two points divides the space into two halves as shown in Fig 4.1(a). The half plane that contains a is denoted as $H_{a:q}$ and the half plane that contains q is denoted as $H_{q:a}$. The perpendicular bisector has the property that any point p (depicted by a star in Fig. 4.1(a)) that lies in $H_{a:q}$ is closer to a than q (i.e., $dist(p, a) \leq dist(p, q)$) and any point y that lies in $H_{q:a}$ is closer to q than a (i.e., $dist(y, q) \leq dist(y, a)$). Hence, q cannot be the closest facility of any point p that lies in $H_{a:q}$, i.e., C_p contains at least one facility a. We say that the point p is pruned by the bisector $B_{a:q}$ if p lies in $H_{a:q}$. Alternatively, we say that the point a prunes the point p. In general, if a point p is pruned by at least k bisectors then C_p contains at least k facilities (i.e., $|C_p| \geq k$).

Existing work [TPL04, WYCT08b, CLZ+09] use this observation to prune the space that cannot contain any RkNN of q. More specifically, an area can be pruned if at least k bisectors prune it. In Fig. 4.1, five facility points (q, a, b, c and d) are shown. In

Fig. 4.1(a) the bisectors between q and two facility points a and b are drawn (see $B_{a:q}$ and $B_{b:q}$). If k is 2, then the white area can be pruned because it lies in two half-planes ($H_{a:q}$ and $H_{b:q}$) and $|C_{p'}| \geq 2$ for any point p' in it. The area that is not pruned is called unpruned area and is shown shaded.



(a) Unpruned area is not influence zone (b) Unpruned area is influence zone

Figure 4.1: Computing influence zone Z_k ($k = 2$)

Although it can be guaranteed that for every point p' in the pruned area $|C_{p'}| \geq k$, it cannot be guaranteed that for every point p in the unpruned area $|C_p| < k$ if we only consider a subset of the bisectors instead of all bisectors. In other words, the unpruned area is not the influence zone. For example, in Fig. 4.1(a), the point p lies in the unpruned area but $|C_p| = 2$ (i.e., C_p contains a and c). Hence, the shaded area of Fig. 4.1(a) is not the influence zone.

One straight forward approach to compute the influence zone is to consider the bisectors of q with every facility point f. If the bisectors of q and all facilities are considered, then the unpruned area is the area that is pruned by less than k bisectors. Fig. 4.1(b) shows the unpruned area (the shaded polygon) after the bisectors $B_{c:q}$ and $B_{d:q}$ are also considered. It can be verified that the shaded area is the influence zone (i.e., for every p in the shaded area $|C_p| < 2$ and for every p' outside it $|C_{p'}| \geq 2$).

However, this straight forward approach is too expensive because it requires computing the bisectors between q and all facility points. We note that for some facilities, we do not need to consider their bisectors. In Fig. 4.1(b), it can be seen that the bisector $B_{d:q}$ (shown in broken line) does not affect the unpruned area (shown shaded). In other words, if the bisectors of a, b and c are considered then the bisector $B_{d:q}$ does not prune more area. Hence, even if $B_{d:q}$ is ignored, the influence zone can be computed.

Next, we present some lemmas that help us in identifying the facilities that can be ignored. Without loss of generality, we assume that the data universe is bounded by a square. Since we use bisectors to prune the space, the unpruned area is always a polygon and is interchangeably called unpruned polygon hereafter. Below we present several lemmas that not only guide us to the final lemma but also help us in few other proofs in the chapter.

Lemma 4.3.1 *A facility f can be ignored if, for* every *point p of the unpruned polygon, the facility f lies outside C_p.*

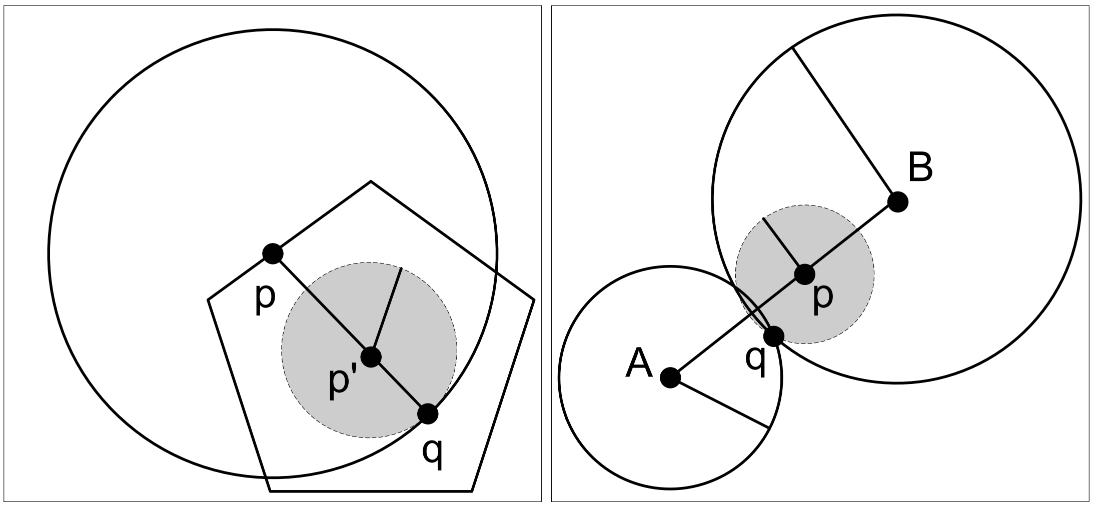
Proof As described earlier, a point p can be pruned by the bisector $B_{f:q}$ iff $dist(p, f) < dist(p, q)$. In other words, the point p can be pruned iff C_p contains f. Hence, if f lies outside C_p, it cannot prune p. If f lies outside C_p for *every* point p, it cannot prune *any* point of the unpruned polygon and can be ignored for this reason. ∎

Checking containment of f in C_p for every point p is not feasible. In next few lemmas, we simplify the procedure to check if a facility point can be ignored.

Lemma 4.3.2 *Let pq be a line segment between two points q and p. Let p' be a point on pq. The circle $C_{p'}$ is contained by the circle C_p.*

Fig. 4.2(a) shows an example where the circle $C_{p'}$ (the shaded circle) is contained by C_p (the large circle). The proof is straight forward and is omitted. Based on this lemma, we present our next lemma.

Lemma 4.3.3 *A facility f can be ignored if, for* every *point p on the* boundary *of the unpruned polygon, f lies outside C_p.*



(a) Lemma 4.3.2 and 4.3.3 (b) Lemma 4.3.4

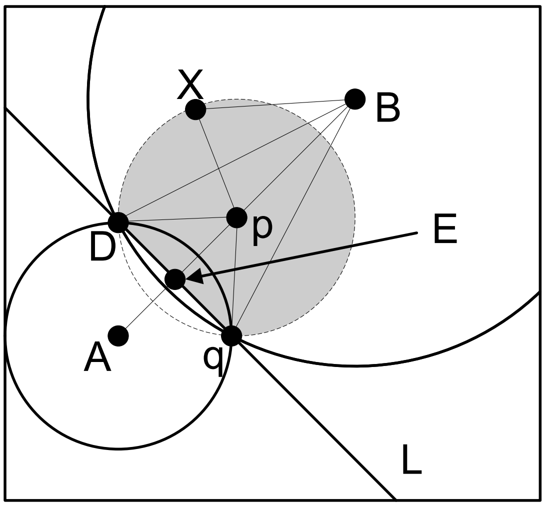
Figure 4.2: Illustration of Lemmas 4.3.2, 4.3.3 and 4.3.4

Proof We prove the lemma by showing that we do not need to check containment of f in $C_{p'}$ for any point p' that lies within the polygon. Let p' be a point that lies within the polygon. We draw a line that passes through q and p' and cuts the polygon at a point p (see Fig. 4.2(a)). From Lemma 4.3.2, we know that C_p contains $C_{p'}$. Hence, if f lies outside C_p, then it also lies outside $C_{p'}$. Hence, it suffices to check the containment of f in C_p for every point p on the boundary of the polygon. ∎

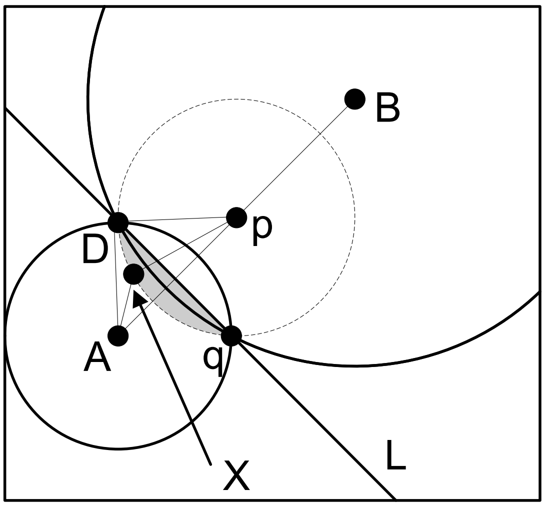
The next two lemmas show that we can check if a facility f can be ignored or not by only checking the containment of f in C_v for every vertex v of the unpruned polygon.

Lemma 4.3.4 *Given a line segment AB and a point p on AB. The circle C_p is contained by $C_A \cup C_B$, i.e., every point in the circle C_p is either contained by C_A or by C_B (see Fig. 4.2(b)).*

Proof Fig. 4.3 shows the line segment AB and the point p. It suffices to show that the boundary of C_p is contained by $C_A \cup C_B$. If q lies on AB, the lemma can be proved by Lemma 4.3.2. Otherwise, we identify a point D such that AB is a segment of the perpendicular bisector between D and q. Then, we draw a line L that passes through points D and q. First, we show that the part of the circle C_p that lies on the right side of L (i.e., the shaded part in Fig. 4.3(a)) is contained by C_B. Then, we show that the part of the circle C_p that lies on the left side of L (i.e., the shaded part in Fig. 4.3(b)) is contained by C_A.



(a) For right side of L



(b) For left side of L

Figure 4.3: Proof of Lemma 4.3.4

We can find the length of qB (denoted as $\overline{qB}$) by using the triangle $\triangle qpB$ and applying the law of cosines (see Fig. 4.3(a)).

$$\overline{qB} = \sqrt{(\overline{pB})^2 + (\overline{pq})^2 - 2 \cdot \overline{pB} \cdot \overline{pq}(Cos\measuredangle Bpq)} \tag{4.1}$$

For any point X that lies on the boundary of C_p and is on the right side of L (i.e., the boundary of the shaded circle in Fig. 4.3(a)), consider the triangle $\triangle\ pXB$. The length of BX can be computed using the law of cosines.

$$\overline{BX} = \sqrt{(\overline{pB})^2 + (\overline{pX})^2 - 2 \cdot \overline{pB} \cdot \overline{pX}(Cos\measuredangle BpX)} \tag{4.2}$$

Please note that the triangles $\triangle qpB$ and $\triangle DpB$ are similar because $\overline{Dp} = \overline{qp}$ and $\overline{DB} = \overline{qB}$ (any point on a perpendicular bisector $B_{u:v}$ is equi-distant from u and v). Due to similarity of triangles $\triangle qpB$ and $\triangle DpB$, $\measuredangle Bpq = \measuredangle BpD$.

It can be shown that $\overline{BX} \leq \overline{qB}$ by comparing Eq. (4.1) and Eq. (4.2). This is because $\overline{pX} = \overline{pq}$ and $\measuredangle BpX \leq (\measuredangle Bpq = \measuredangle BpD)$. Since cosine monotonically decreases as the angle increases from 0° to 180°, $\overline{BX} \leq \overline{qB}$. This means the point X lies within the circle C_B.

Similarly, for any X that lies on the part of circle C_p that is on left side of the line L (see Fig. 4.3(b)) it can be shown that $\overline{AX} \leq (\overline{AD} = \overline{Aq})$. This can be achieved by considering the triangles $\triangle pXA$ and $\triangle pDA$ and using law of cosines to obtain $\overline{AX}$ and $\overline{AD}$ (the key observation is that $\measuredangle XpA \leq \measuredangle DpA$). ∎

Lemma 4.3.5 *A facility f can be ignored if, for every vertex v of the unpruned polygon, the facility f lies outside C_v.*

Proof Let AB be an edge of the polygon. From Lemma 4.3.4, we know that if a facility f lies outside C_A and C_B, then it lies outside C_p for every point p on the edge AB. This implies that if f lies outside C_v for every vertex v of the polygon then it lies outside C_p for every point p that lies on the boundary of the polygon. Such facility f can be ignored as stated in Lemma 4.3.3. ∎

Next lemma shows that we only need to check this condition for *convex* vertices. First, we define the convex vertices.

Definition 4.3.6 *Consider a polygon P where V is the set of its vertices. Let H_{con} be the convex hull of V. The vertices of H_{con} are called convex vertices of the polygon P and the set of the convex vertices is denoted as V_{con}.*

Fig. 4.4 shows an example where a polygon with vertices A to J is shown in broken lines. Its convex hull is shown in solid lines which contains the vertices A, C, E, G and I and these vertices are the convex vertices. Note that $V_{con} \subseteq V$.

Lemma 4.3.7 *A facility f can be ignored if it lies outside C_v for every* convex vertex *v of the unpruned polygon P.*

Proof By definition of a convex hull, the convex hull H_{con} contains the polygon P. If a facility point f does not prune any point of the convex polygon H_{con}, it cannot prune any point of the polygon P because $P \subseteq H_{con}$. Hence, it suffices to check if f prunes any point of H_{con} or not. From Lemma 4.3.5, we know that f does not prune any point of H_{con} if it lies outside C_v for every vertex v of H_{con}. Hence, f can be ignored if it lies outside every C_v where v is a vertex of the convex polygon (i.e., v is a convex vertex). ∎

The above lemma identifies a condition for a facility f to be ignored. Next lemma shows that any facility that does not satisfy this condition prunes at least one point of the unpruned area. In other words, next lemma shows that the above condition is tight.

Lemma 4.3.8 *If a facility f lies in* any *C_v for any convex vertex v of the unpruned polygon P then there exists at least one point p in the polygon P that is pruned by f.*

Proof If f lies in C_v for any $v \in V_{con}$, it means that $dist(f, v) < dist(f, q)$. Hence, f prunes the vertex v. Since $V_{con} \subseteq V$, the vertex v is a point in the polygon P. ∎

4.3.2 Algorithm

Based on the problem characteristics we described earlier in this section, we propose an algorithm to efficiently compute the influence zone. We assume that the facilities are indexed by an R-tree [Gut84]. The main idea is that the facilities are iteratively retrieved and the space is iteratively pruned by considering their bisectors with q. The facilities that are close to the query q are expected to prune larger area and are given priority.

Algorithm 5 presents the details. Initially, the whole data space is considered as the influence zone and the root of the R-tree is inserted in a min-heap h. The entries are iteratively de-heaped from the heap. The entries in the heap may be rectangles (e.g., intermediate nodes) or points. If a de-heaped entry e completely lies outside C_v of all convex vertices of the current influence zone (e.g., the current unpruned area), it can be ignored. Otherwise, it is considered valid (lines 5 to 7). If the entry is valid and is an intermediate node or a leaf node, its children are inserted in the heap (lines 8 to 10). Otherwise, if the entry e is valid and is a data object (e.g., a facility point), it is used to prune the space. The current influence zone is also updated accordingly (line 12). The algorithm stops when the heap becomes empty.

Algorithm 5 Compute Influence Zone

```
Input: a set of objects $O$, a query $q \in O$, $k$
Output: Influence Zone $Z_k$
1: initialize $Z_k$ to the boundary of data universe
2: insert root of R-tree in a min-heap $h$
3: while $h$ is not empty do
4:    deheap an entry $e$
5:    for each convex vertex $v$ of $Z_k$ do
6:       if $mindist(v, e) < dist(v, q)$ then
7:          mark $e$ as valid; break
8:    if $e$ is valid then
9:       if $e$ is an intermediate node or leaf then
10:         insert every child $c$ in $h$ with key $mindist(q, c)$
11:      else if $e$ is an object then
12:         update the influence zone $Z_k$ using $e$
```

The proof of correctness follows from the lemmas presented in the previous section because only the objects that do not affect the unpruned area are ignored. It is also important to note that the entries of R-tree are accessed in ascending order of their minimum distances to the query. The nearby facility points are accessed and the unpruned area keeps shrinking which results in a greater number of upcoming entries being pruned. Hence, the entries that are far from the query are never accessed.

Updating unpruned polygon

Now, we briefly describe how to update the unpruned polygon (or current influence zone) when a new facility point f is considered (line 12 of Algorithm 5). The main idea is similar to [WYCT08b]. The intersection points between all the bisectors are maintained. Each intersection point is assigned a counter that denotes the number of bisectors that prune it. Fig. 4.5 shows an example ($k = 2$) where three bisectors $B_{a:q}$, $B_{b:q}$ and $B_{c:q}$ have been considered. The counter of intersection point v_{11} is 2 because it is pruned by $B_{b:q}$ and $B_{c:q}$. The counter of v_8 is 1 because it is pruned only by $B_{c:q}$. It can be immediately verified that the unpruned area can be defined by only the intersection points with counters less than k [WYCT08b] (see the shaded area of Fig. 4.5). Hence, we can discard the intersection points with counters at least equal to k.

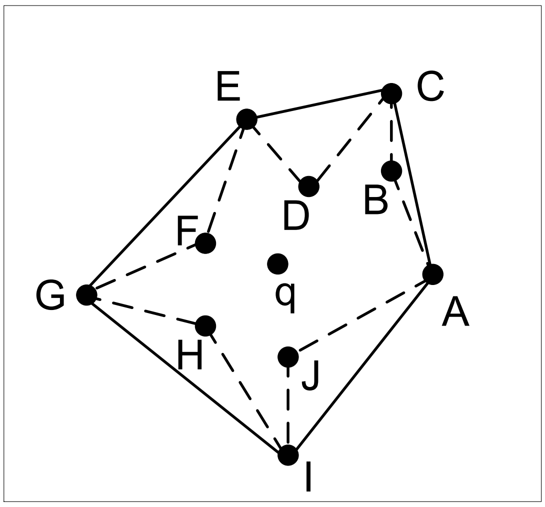


Figure 4.4: Convex polygon

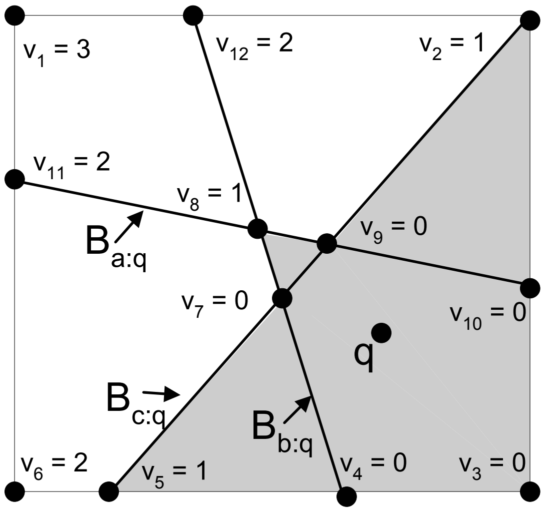


Figure 4.5: Computing counters

Algorithm 6 shows the details of updating the influence zone when a new facility f is considered. Firstly, the algorithm computes the new intersection points between $B_{f:q}$ and the existing bisectors. The counters of these new intersection points are also computed (line 1). Then, the algorithm updates the counters of all existing intersection points (line 2). More specifically, the counter of an existing intersection point p is incremented by one if $B_{f:q}$ prunes p. Otherwise, the counter remains unchanged. The algorithm discards the intersection points with counters at least equal to k (line 3). Then, the algorithm computes the current unpruned polygon and determines the convex vertices (lines 4 and 5). Recall that determining the convex vertices is important in order to apply Lemma 4.3.7.

We remark that the first three lines of Algorithm 6 are the same as used in the

Algorithm 6 update influence zone

```
Input: current influence zone $Z_k$, a new facility $f$
Output: updated influence zone $Z_k$
1: compute new intersection points and their counters
2: update the counters of existing intersection points
3: discard intersection points with counters at least equal to $k$
4: compute the unpruned polygon
5: find the convex vertices
```

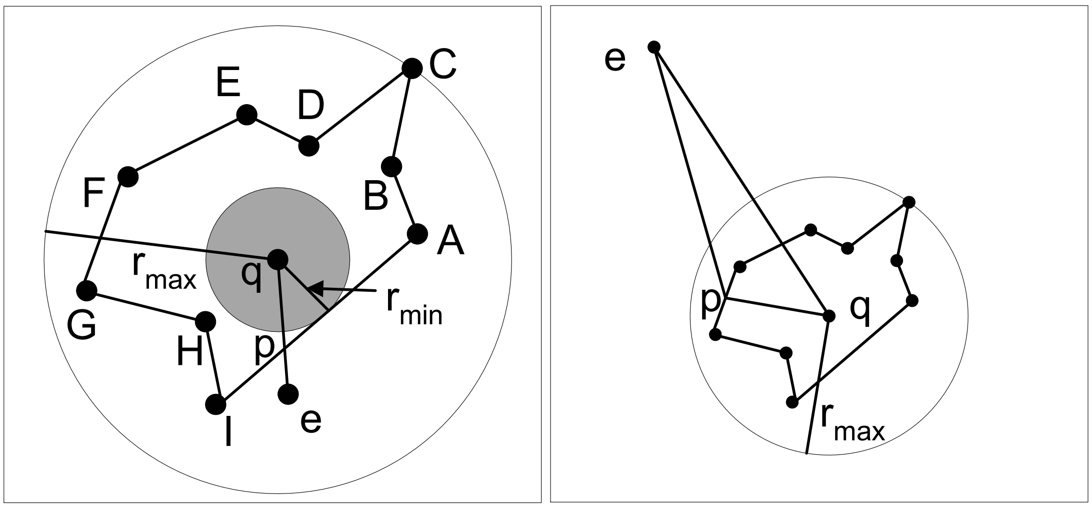
technique proposed by Wu et. al [WYCT08b]. They showed by a simple analysis that the complexity of these lines is $O(m^2)$ where m is the number of existing bisectors contributing to the unpruned polygon. Later in Section 4.5.4, we conduct a more rigorous complexity analysis and show that the overall complexity of Algorithm 6 can be reduced to $O(km)$ when k is smaller than m.

Optimizations

In this section, we present few optimizations to improve the efficiency of Algorithm 5. It can be shown that the number of convex vertices is $O(m)$ where m is the number of bisectors considered so far [WYCT08b] (i.e., m is the number of facilities used to update the current influence zone at line 12 of Algorithm 5). Hence, checking whether an entry of the R-tree is valid or not requires $O(m)$ distance computations (see lines 5- 7 of Algorithm 5). Next, we present few observations and show that we can determine the validity of some entries by a single distance computation.

Lemma 4.3.9 *Let r_{min} be the minimum distance of q to the boundary of the unpruned polygon. Then, an entry e is a valid entry if $mindist(q, e) < 2r_{min}$ (Fig. 4.6(a) shows r_{min}).*

Proof To prove that e is a valid entry, we show that there exists at least one point p in the unpruned polygon such that C_p contains e. If e lies inside the unpruned polygon then



(a) Lemma 4.3.9 (b) Lemma 4.3.10

Figure 4.6: Optimizations

e is a valid entry because C_e contains e and e is a point in the unpruned polygon. Now, we prove the lemma for the case when e lies outside the unpruned polygon. Fig. 4.6(a) shows an entry e for which $dist(q, e) < 2r_{min}$. We draw a line that passes through e and q and intersects the boundary of the unpruned polygon at a point p. Clearly, $dist(p, e) = dist(q, e) - dist(p, q)$. We know that $dist(q, e) < 2r_{min}$ and $dist(p, q) \geq r_{min}$. Hence, $dist(p, e) \leq r_{min}$ which implies that $dist(p, e) \leq dist(p, q)$. Hence, e lies in C_p.
■

Lemma 4.3.10 *Let r_{max} be the distance of q to the furthest vertex of the unpruned polygon. Then, an entry e of the R-tree is an invalid entry if $mindist(e, q) > 2r_{max}$.*

Proof Fig. 4.6(b) shows r_{max} and a point e such that $dist(e, q) > 2r_{max}$. Consider a point p on the boundary of the unpruned polygon. By the definition of r_{max}, $dist(p, q) \leq r_{max}$. Clearly, $dist(p, q) + dist(p, e) \geq dist(q, e)$ (this covers both the cases when p lies on the line qe and when $\triangle qpe$ is a triangle). Since, $dist(p, q) \leq r_{max}$ and $dist(e, q) > 2r_{max}$, $dist(p, e)$ must be greater than r_{max}. Hence, $dist(p, e) > dist(p, q)$ which means e lies outside C_p. This holds true for every point p on the boundary of the unpruned polygon. Hence, e can be ignored (i.e., e is invalid). ■

If an entry of the R-tree satisfies one of the above two lemmas, we can determine its validity without computing its distances from the convex vertices. Note that r_{max} and r_{min} can be computed in linear time to the number of edges of the unpruned polygon and are only computed when the influence zone is updated at line 12 of Algorithm 5.

4.3.3 Checking containment in the influence zone

The applications that use influence zone may require to frequently check if a point or a shape lies within the influence zone or not. Although the suitability of a method to check the containment depends on the nature of the application, we briefly describe few approaches.

One simple approach is to record all the objects that were accessed during the construction of the influence zone (the objects for which the bisectors were considered). If a shape is pruned by less than k of these bisectors then the shape lies inside the influence zone otherwise it lies outside the influence zone. This approach takes linear time in number of the accessed objects. Moreover, checking whether a point is pruned by a bisector $B_{f:q}$ is easy (e.g., if $dist(p, f) < dist(p, q)$ then the point p is pruned otherwise not). Hence, a point containment check requires $O(m)$ distance computations where m is the number of the accessed objects.

Before we show that the point containment can be done in logarithmic time, we define a *star-shaped polygon* [PS85]. A polygon is a star-shaped polygon if there exists a point z in it such that for each point p in the polygon the segment zp lies entirely in the polygon. The point z is called a kernel point. The polygon shown in Fig. 4.6(a) is a star-shaped polygon and q is its kernel point. Fig. 4.7(a) shows a polygon that is not star-shaped (the segment qp does not lie entirely in the polygon). Let n be the number of vertices of a star-shaped polygon. After a linear time pre-processing, every point containment check can be done in $O(log\ n)$ if a kernel point of the polygon is known [PS85]. Please see [PS85] for more details.

Lemma 4.3.11 *The influence zone is always a star-shaped polygon and q is its kernel point.*

Proof We prove this by contradiction. Assume that there is a point p in the influence zone such that the segment pq does not lie completely within the influence zone. Fig. 4.7(a) shows an example, where a point p' lies on the segment pq but does not lie within the influence zone. From Lemma 4.3.2, we know that C_p contains $C_{p'}$. Since p' is a point outside the influence zone, $|C_{p'}| \geq k$. As $C_{p'}$ is contained by C_p, $|C_p| \geq k$. Hence, p cannot be a point inside the influence zone which contradicts the assumption. ∎

Since the maximum number of vertices of the influence zone is $O(m^2)$, the point containment check can be done in $O(log\ m)$. Next, we present two simple checks to reduce the cost of containment check in certain cases by using r_{max} and r_{min} we introduced earlier.

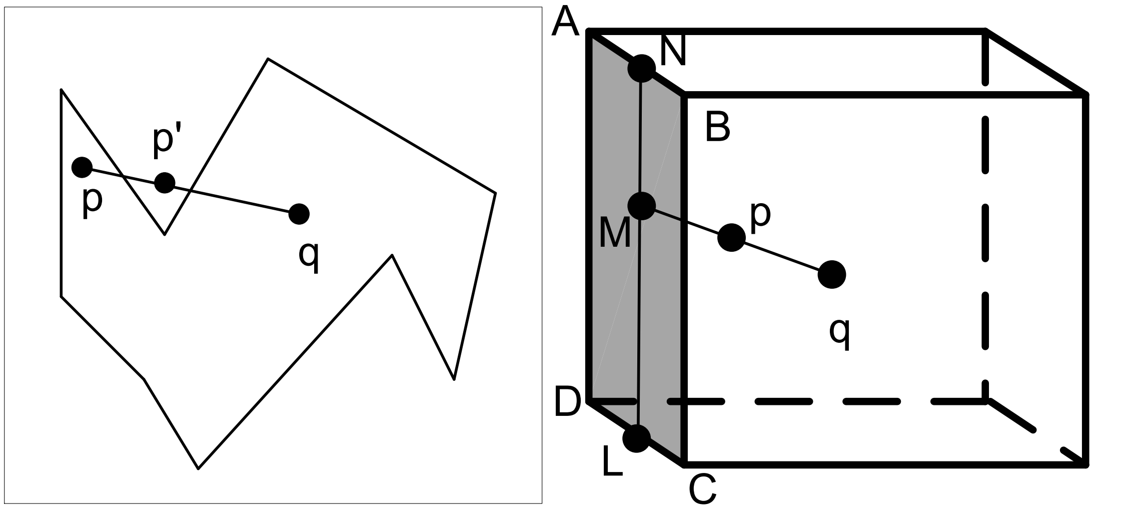
Let r_{min} and r_{max} be as defined in Lemma 4.3.9 and 4.3.10, respectively. Then, the circle centered at q with radius r_{max} (the big circle in Fig. 4.6(a)) completely contains the influence zone. Similarly, the circle centered at q with radius r_{min} (the shaded circle in Fig. 4.6(a)) is completely contained by the influence zone. Hence, any point p that has a distance greater than r_{max} from q is not contained by the influence zone and any point p' that lies within distance r_{min} of q is contained by the influence zone.

For the applications that allow relatively expensive pre-processing, the influence zone can be indexed (e.g., by a grid or a quad-tree) to efficiently check the containment. For example, for the continuous monitoring of RkNN queries, we use a grid to index the influence zone. The details are presented in next section.

4.3.4 Extension to higher dimensions

In dimensions higher than two, the bisectors are called half spaces and the unpruned region is a polyhedron instead of a polygon [OBSC99]. The circle C_p centered at p with radius $dist(p, q)$ is called a hypersphere. It can be shown that Lemma 4.3.4 holds for higher dimensions. This can be proved by a projection on a two dimensional space for each point of the hypersphere.

The space is pruned in a similar way as in two dimensional space, i.e., the space that is pruned by at least k half spaces is pruned. The following lemma holds for the unpruned area which is a polyhedron.



(a) Lemma 4.3.11 (b) Lemma 4.3.12

Figure 4.7: Lemmas 4.3.11 and 4.3.12

Lemma 4.3.12 *A facility point f can be ignored if, for every vertex v of the unpruned polyhedron, f lies outside C_v.*

Proof We prove the lemma for a 3-dimensional polyhedron and the proof for the arbitrary dimensionality is similar. Let p be any point inside the polyhedron as shown in Fig. 4.7(b). We draw a line that passes through p and q and crosses a face (the shaded face $ABCD$) of the polyhedron at a point M. For such point M, we can always draw a line on this face of the polyhedron such that it passes through M and intersects the edges of the face at points L and N as shown in Fig 4.7(b). From Lemma 4.3.4, C_A and C_B contain C_N. Similarly, C_C and C_D contain C_L. Again, from Lemma 4.3.4, C_N and C_L contain C_M. Lastly, C_M contains C_p (Lemma 4.3.2). Hence, C_p is contained by the hyperspheres of the vertices of the face $ABCD$ (C_A, C_B, C_C and C_D). This holds for any arbitrary point p inside the polyhedron. Hence, we only need to check the containment in C_v for every vertex v of the polyhedron. ∎

Given Lemma 4.3.12, it can be immediately verified that Lemmas 4.3.7 and 4.3.8 also hold in dimensions higher than two.

4.4 Applications in RkNN Processing

4.4.1 Snapshot Bichromatic RkNN Queries

Our algorithm consists of two phases namely *pruning* phase and *containment* phase.

Pruning Phase. In this phase, the influence zone Z_k is computed using the given set of facilities.

Containment Phase. By the definition of influence zone Z_k, a user u can be the bichromatic RkNN if and only if it lies within the influence zone Z_k. We assume that the set of users are indexed by an R-tree. The R-tree is traversed and the entries that lie outside the influence zone are pruned. The objects that lie in the influence zone are RkNNs.

4.4.2 Snapshot Monochromatic RkNN Queries

By definition of a monochromatic RkNN query (see Section 4.2), a facility f is the RkNN iff $|C_f| < k + 1$. Hence, a facility that lies in Z_{k+1} is the monochromatic RkNN of q where Z_{k+1} is the influence zone computed by setting k to $k + 1$. Below, we highlight our technique.

Pruning Phase. In this phase, we compute the influence zone Z_{k+1} using the given set of facilities F. We also record the facility points that are accessed during the construction of the influence zone and call them the candidate objects.

Containment Phase. Please note that every facility point that is contained in the influence zone Z_{k+1} will be accessed during the pruning phase. This is because every facility that lies in the influence zone cannot be ignored during the construction of the influence zone (inferred from Lemma 4.3.1). Hence, the set of candidate object contains all possible RkNNs. For each of the candidate object, we report it as RkNN if it lies within the influence zone Z_{k+1}.

4.4.3 Continuous monitoring of RkNNs

In this section, we present our technique to continuously monitor bichromatic RkNN queries (see the problem definition in Section 4.2). The basic idea is to index the influence zone by a grid. Then, the RkNNs can be monitored by tracking the users that enter or leave the influence zone.

Initially, the influence zone Z_k of a query q is computed by using the set of facility points. We use a grid based data structure to index the influence zone. More specifically, a cell c of the grid is marked as an *interior* cell if it is completely contained by the influence zone. A cell c' is marked as a *border* cell if it overlaps with the boundary of the influence zone. Fig. 4.8 shows an example where the influence zone is the polygon $ABCDFEGHI$, interior cells are shown in dark shade and the border cells are the light shaded cells.

For each border cell, we record the edges of the polygon that intersect it. For example, in c_1, we record the edge AI and in c_2 we record the edges AI and HI. If a user $u \in U$ is in an interior cell, we report it as RkNN of the query. If a user lies in a border cell, we check if it lies outside the polygon by checking the edges stored in this cell. For example, if a user lies in c_1 and it lies inside AI, we report it as RkNN.

4.5 Theoretical Analysis

We assume that the facilities and the users are uniformly distributed in a unit space. The number of facilities is $|F|$. For bichromatic queries, the number of users is $|U|$.

4.5.1 Area of Influence Zone

Before we analyse the area of the influence zone, we show the relationship between an order k Voronoi cell and the influence zone. We utilize this relationship to analyse the area of the influence zone.

Relationship with order k Voronoi cell: An order k Voronoi diagram divides the space into cells and we refer to each cell as a k-Voronoi cell. Each k-Voronoi cell is related to a set of k facility points (denoted as F_k) such that for any point p in this cell the k closest facilities are F_k. Fig. 4.9 shows an order 2 Voronoi diagram computed on the facility points a to i. Each cell c is related to two facility points (shown as $\{f_i, f_j\}$ in Fig. 4.9) and these are the two closest facilities for any point p in c. For example, for any point p in the cell marked as $\{a, e\}$ the two closest facilities are a and e.

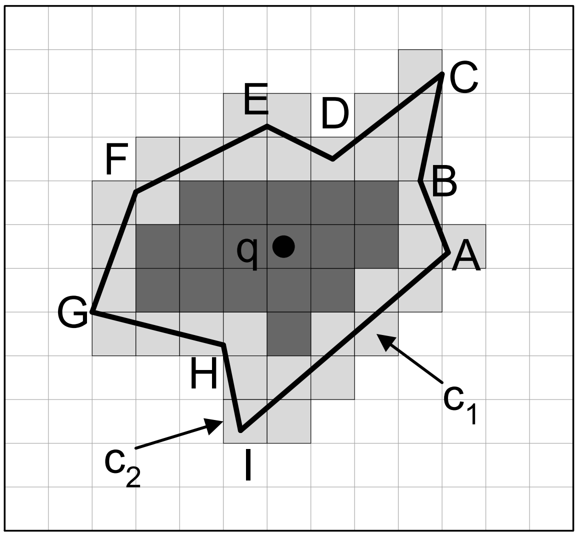


Figure 4.8: Continuous monitoring

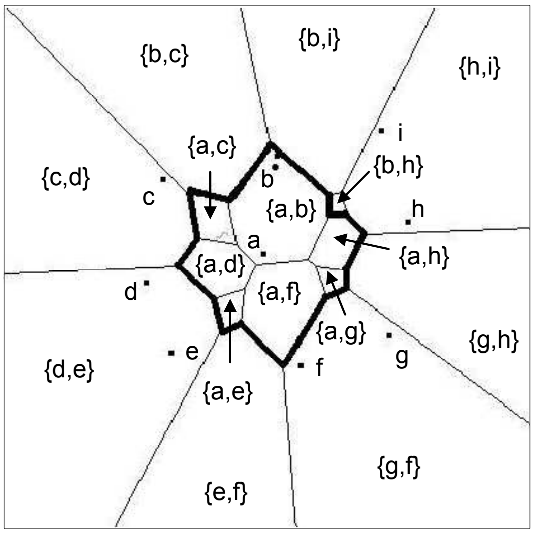


Figure 4.9: Order 2 Voronoi diagram

Clearly, when $k = 1$ the k-Voronoi cell related to q is exactly the same as the influence zone. For $k > 1$, the influence zone corresponds to the union of all k-Voronoi cells that are related to q (i.e., have q in their F_k). For example, in Fig. 4.9, the influence zone of the facility a is shown in bold boundary and it corresponds to the union of the cells related to a.

Now, we analyse the area of the influence zone.

Consider the influence zones of all the facilities in the data set. Every point in the unit space lies in a cell that is related to k facilities. This implies that every point lies in exactly k influence zones (e.g., in Fig. 4.9, every point in the cell marked as $\{a, f\}$ lies in the influence zone of a as well as the influence zone of f). Hence, the sum of the areas of all the influence zones is k. Since the total number of facility points is $|F|$, the expected area of a randomly chosen facility point is $k/|F|$.

Remark: The above discussion shows that the influence zone can be computed by using a pre-computed order k Voronoi diagram. However, as mentioned in [ZZP+03], a technique that uses a pre-computed order k Voronoi diagram may not be practical for the following reasons : i) the value of k may not be known in advance; ii) even if k is known in advance, order k Voronoi diagrams are very expensive to compute and incur high space requirement; iii) spatial indexes are useful for all query types and pre-computed Voronoi diagrams may not be used for all queries. In contrast, R-tree based indexes used by our algorithm are used for many important queries.

4.5.2 Number of RkNNs

First, we evaluate the number of bichromatic RkNNs. We assume that the users are uniformly distributed in the space. The number of users that lie in the influence zone is the number of bichromatic RkNNs. Hence, the number of bichromatic RkNNs is $|U|.k/|F|$.

The area of the influence zone Z_{k+1} for a monochromatic RkNN query is $(k+1)/|F|$. The number of facilities in this area is $(k + 1)$ which includes the query. Hence the expected number of monochromatic RkNNs is k.

4.5.3 IO cost of our algorithms

Before we analyse the IO costs of our proposed algorithms, we analyse the cost of a *circular range* query. Then, we analyse the costs of our algorithms by using the IO cost of the circular range queries.

IO cost of a circular range query: A circular range query [CBL+10] finds the objects that lie within distance r of the query location. We assume that the objects are indexed by an R-tree and analyse the number of nodes that lie within the range of the query. Fig. 4.10(a) shows a circular range query where the search area is the circle centered at q with radius r (the shaded circle). The approach to analyse the IO cost of the circular range query is similar to the IO cost analysis of window queries presented in [TSS00]. Let R_l be the number of rectangles at level l of the R-tree. Let s_l be the side length of each rectangle at level l (the rectangles of a good R-tree have similar sizes [KF93]). We assume that the centers of rectangles at each level follow a uniform distribution. Let d_l be the diagonal length of each rectangle at level l. As shown in Fig. 4.10(a), any rectangle that has its center c at a distance greater than $r + d_l/2$ cannot intersect the range query and should not be accessed. Hence, the number of rectangles (nodes) accessed at level l is $\pi(r + d_l/2)^2 R_l$ which is the number of center points c that lie in the circle of radius $r + d_l/2$ (the large circle in Fig. 4.10(a)).

Now, we need to compute d_l and R_l for each level l. Let S be the number of objects indexed by the R-tree. Let f be the fanout of the tree. The number of rectangles R_l at level l of the R-tree is S/f^l (e.g., leaf nodes are at level 1 and the number of leaf level rectangles is S/f). Since we assume uniform distribution of points, each rectangle at level l contains f^l points. In other words, the area of each rectangle is f^l/S. Assuming that the both sides of a rectangle are of same size, the side length s_l is $\sqrt{f^l/S}$. Given s_l, half of the diagonal length $d_l/2$ can be computed easily which is $\sqrt{f^l/2S}$.

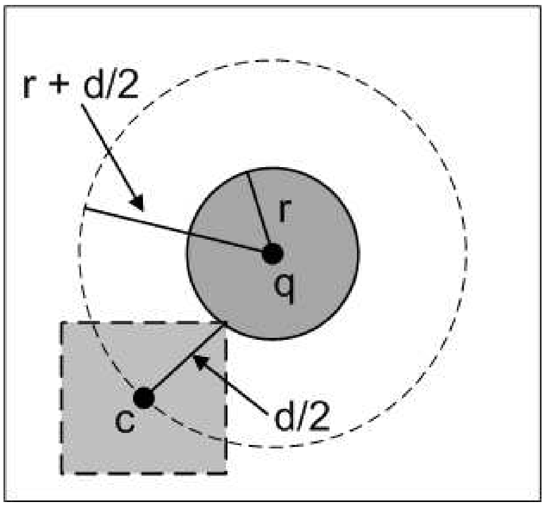
The total IO cost (the total number of nodes accessed) is obtained by applying the formula for each level l. The total number of levels excluding the root is $\lfloor log_f S \rfloor$. The root is accessed anyway, so one is added to this cost. Hence, the total IO cost is obtained by the following equation.

$$Range\ query\ cost = 1 + \sum_{l=1}^{\lfloor log_f S \rfloor} \pi(r + \sqrt{f^l/2S})^2 S/f^l \tag{4.3}$$

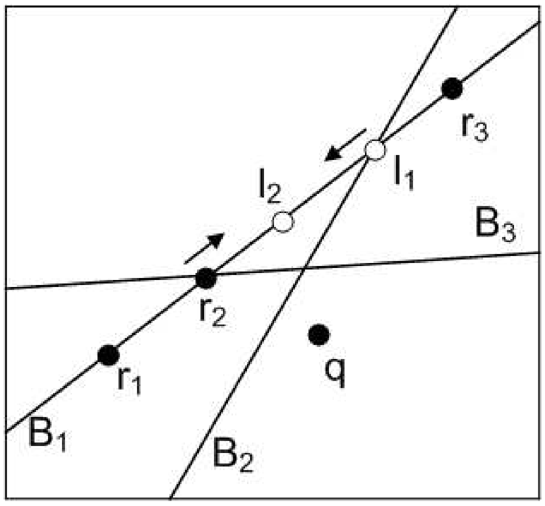
Based on this, first we analyse the cost of computing the influence zone and then we analyse the costs of our RkNN algorithms.

IO cost of computing the influence zone: We approximate the influence zone to a circular shape having the same area (we noted that as k gets larger the shape of influence zone has more resemblance with a circle). Since the area of the influence zone Z_k is $k/|F|$, the radius of the circle can be computed as $r_k = \sqrt{\frac{k}{\pi|F|}}$. From Lemma 4.3.5, an object can be ignored if it lies at a distance greater than $dist(q, v)$ from every vertex v of the unpruned area. Since we assume that each vertex is at same distance r_k from the query (i.e., influence zone is a circle), an object can be ignored if it lies at a distance greater than $2r_k$ from q. Hence, the objects within the range $2r_k$ of the query are accessed during the computation of the influence zone. The IO cost can be found by replacing r in Eq. (4.3) with $2r_k = 2\sqrt{\frac{k}{\pi|F|}}$ and S with $|F|$ (the number of the facility points).

IO cost of a monochromatic RkNN query: The IO cost for monochromatic RkNN query is the same as the IO cost of computing the influence zone Z_{k+1}. This is because the R-tree is traversed only during the construction of the influence zone (i.e., the containment phase does not access R-tree). Hence, IO cost can be found by replacing r in Eq. (4.3) with $2r_{k+1} = 2\sqrt{\frac{k+1}{\pi|F|}}$ and S with $|F|$.



(a) Range query



(b) Lemma 4.5.1

Figure 4.10: Illustration of theoretical analysis

IO cost of a bichromatic RkNN query: The cost of the pruning phase is the same as the cost of computing the influence zone Z_k which we have computed earlier. The cost of the containment phase is the cost of accessing the users that lie within the influence zone which can be computed in a similar way. More specifically, only the users that lie within distance r_k (the radius of the influence zone) of q are accessed. Hence, the cost of the containment phase can be computed by replacing r in Eq. (4.3) with $r_k = \sqrt{\frac{k}{\pi|F|}}$ and S with $|U|$ where $|U|$ is the number of users indexed by the R-tree.

4.5.4 Complexity Analysis

In this section, we show that the complexity of Algorithm 6 (if implemented properly) is $O(km)$ where m is the number of facilities (or bisectors) considered so far. Recall that Algorithm 6 discards every intersection point having counter at least equal to k (see line 3). Throughout this section, any intersection point that has a counter less than k is called a valid intersection point.

For each line of Algorithm 6, we show that its complexity is at most $O(km)$. For the ease of the presentation, we analyse the complexity of these lines in an order different from the order in which they appear in the algorithm.

Complexity of line 2: update the counters of existing intersection points

A simple analysis shows that the total number of intersection points is at most $O(m^2)$. As we only keep the intersection points that are valid, we only need to update the counters of the valid intersection points. Before we show that the number of valid intersection points is $O(km)$, we define few terms and notations.

Consider the example of Fig. 4.10(b) where two bisectors B_2 and B_3 intersect a bisector B_1 at points l_1 and r_2, respectively. The bisector B_2 prunes every point on B_1 that lies on the left side of l_1 (as shown with an arrow). For example, B_2 prunes the points l_2, r_2 and r_1. The intersection point l_1 is called a *left pruning* intersection of B_1 because it prunes every point on B_1 that lies on its left side. The bisector B_3

prunes every point on B_1 that lies on the right side of r_2 (e.g., the points l_2, l_1 and r_3). The intersection point r_2 is called a *right pruning* intersection of B_1. In Fig. 4.10(b), the right pruning intersection points are shown as r_i (black circles) and the left pruning intersection points are shown as l_i (the hollow circles). To keep Fig. 4.10(b) simple, we do not show the bisectors related to r_1, r_3 and l_2.

Note that the counter of any point p on B_1 is at least equal to the number of left pruning intersections on its right side plus the number of right pruning intersections on its left side. For example, the counter of point l_2 is $1 + 2 = 3$ because it is pruned by l_1, r_1 and r_2.

Lemma 4.5.1 shows that each existing bisector can have at most $2k$ valid intersection points.

Lemma 4.5.1 *For any bisector B_1, the number of valid intersection points*[2] *on it is at most $2k$.*

Proof Let the number of right pruning intersection points of B_1 be u. We denote the right pruning intersections of B_1 by $r_1, ..., r_u$ such that for any intersection r_i there are $i - 1$ right pruning intersections on its left. For example, in Fig. 4.10(b), there are two right pruning intersections (r_1 and r_2) on the left side of r_3. For any right pruning intersection point r_i, its counter is at least equal to $i - 1$ because r_i is pruned by at least $i - 1$ right pruning intersections. Hence, only the intersections r_i for $0 < i \leq k$ can have counters less than k. This implies that the number of right pruning intersection points that are valid is at most k. Following similar arguments, it can be shown that at most k left pruning intersection points are valid intersections. Hence, the total number of valid intersection points on B_1 is at most $2k$. ∎

Lemma 4.5.1 shows that each existing bisector can have at most $O(k)$ valid intersection points. Since m is the number of existing bisectors, the total number of valid

[2]The proof of this lemma assumes that each intersection point is unique, i.e., two bisectors do not intersect B_1 at the same point. However, the complexity analysis remains the same even in the absence of this assumption. This is because such intersection points can be merged and treated as one intersection point because their counters would be exactly the same.

intersection points is $O(km)$. Recall that, to update the counter of an intersection point p, we only need to check whether it is pruned by $B_{f:q}$ or not where f is the new facility being considered. This can be done in constant time. Hence, the complexity of line 2 of Algorithm 6 is $O(km)$.

Complexity of line 5: find the convex vertices

We show that we only need to scan the list of the intersection points once to determine the convex vertices. Since the total number of intersection points is $O(km)$, the complexity of this step is $O(km)$. Lemma 4.3.7 is the key to obtain the required complexity.

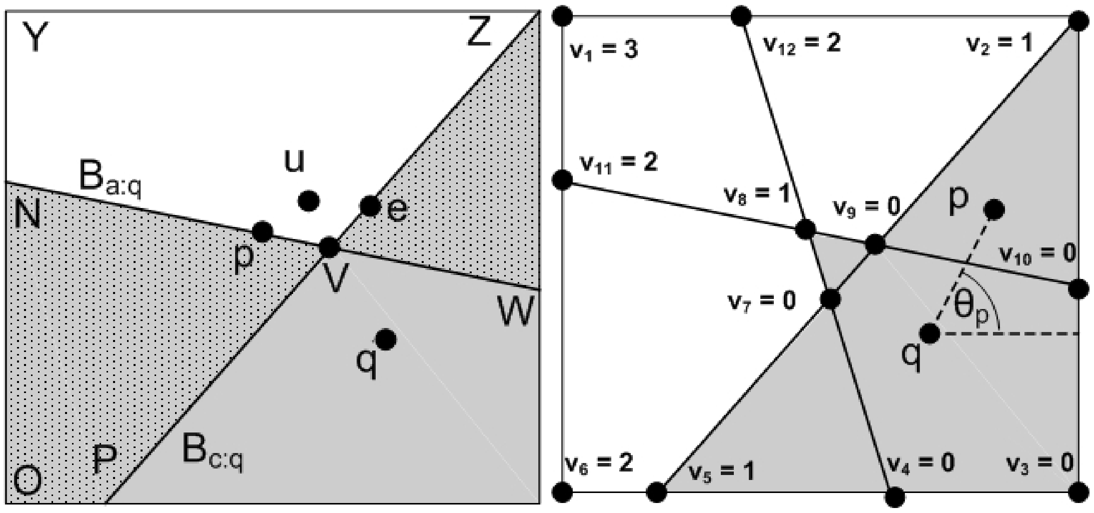
Lemma 4.5.2 *Among the intersection points that do not lie on the boundary of the data universe, only the intersection points with counters equal to $k-1$ can be the convex vertices.*

Proof Any intersection that has a counter greater than $k-1$ is pruned by at least k objects hence cannot be on the boundary of the influence zone (hence, cannot be a convex vertex). Now, we show that the intersections that have counters less than $k-1$ cannot be the convex vertices.

Consider the example of Fig. 4.11(a) where a vertex V has been shown which is the intersection point of two bisectors $B_{a:q}$ and $B_{c:q}$. Suppose that the counter of the vertex V is n. Now, imagine a point p that lies on the line VN and is infinitely close to the vertex V. Clearly, the point p is pruned by at most $n+1$ bisectors[3]. This is because it is pruned by n bisectors that prune V and the bisector $B_{c:q}$. Following the similar argument, we can say that any point e that lies on the line VZ and is infinitely close to V has a counter at most $n+1$. The counter of any point u that lies in the polygon $VNYZ$ (white area) and is infinitely close to V is at least $n+2$ (it is pruned by $B_{c:q}$ and $B_{a:q}$ in addition to all the bisectors that prune V).

[3]In this proof, we assume that only two bisectors pass through the intersection point V. For the special case, when more than two bisectors pass through a vertex V, we may choose to treat V as a convex vertex. Note that this does not affect the correctness of the algorithm because checking containment in a vertex that is not a convex vertex does not affect the correctness.

If the counter n of the vertex V is less than or equal to $k-2$, then the line VN has at least one point p that has counter at most $k-1$ (i.e., $n+1$ as shown above). Hence, the line VN has at least one point p that lies in the influence zone. Similarly, the line VZ has at least one point e that lies in the influence zone. Clearly, the angle eVp is at least $180°$. By definition of a convex hull, no internal angle of a convex hull can be greater than 180^0. Hence, the vertex V is not a convex vertex if its counter is less than or equal to $k-2$. ■



(a) Lemma 4.5.2 (b) Counters

Figure 4.11: Finding convex vertices

In Fig. 4.11(b), we revisit the example of Fig. 4.5. The vertices v_7 and v_9 do not lie on the boundary of the data universe and have counters less than $k-1$ (where $k=2$). Hence, they are not the convex vertices. Among the points that lie on the boundary of the data universe and have counters less than k, only the two extreme points for each boundary line can be the convex vertices. For example, in Fig. 4.5, the lower horizontal boundary line contains 4 vertices (v_3, v_4, v_5 and v_6). The vertex v_6 has counter not less than k and can be ignored. Among the remaining vertices, we consider the extreme vertices (v_3 and v_5) as the convex vertices. Following the above strategy, the convex vertices in Fig. 4.5 are v_3, v_2, v_8 and v_5.

The above discussion shows that the convex vertices can be found by scanning the list of intersection points once. Hence, the cost of finding the convex vertices is $O(km)$.

Complexity of line 4: compute the unpruned polygon

For any point p, we use θ_p to denote the angle formed by the horizontal line passing through q and the line segment pq (see Fig. 4.11(b)). We show that the unpruned polygon can be computed in $O(km)$ if all the intersection points are sorted according to θ_p. Later in Section 4.5.4, we show that we can keep the intersection points sorted in $O(k \cdot log\ m)$ after each call of Algorithm 6. In this section, we assume that the list of intersection points is already sorted according to θ_p.

Lemma 4.5.3 *The unpruned polygon is always a star-shaped polygon and q is its kernel point.*

Proof Consider that $F' \subset F$ is a set of facilities that consist of only the facilities that have been considered so far. Clearly, the current unpruned polygon is the influence zone of q for the data set F'. Hence, Lemma 4.3.7 can be immediately applied to prove that the unpruned polygon is always a star-shaped polygon. ■

Since the unpruned polygon P is a star-shaped polygon and q is its kernel point, every point on its boundary is visible from q [IK95]. This implies that θ_p is unique for every point p on the boundary of P, i.e., $\theta_p \neq \theta_{p'}$ for any two points p and p' that lie on the boundary of P. Hence, given a list of points that lie on the boundary of P, we can construct the polygon P by connecting the points in sorted order of the angles they make with q. Finally, we need to determine the intersection points that lie on the boundary of the unpruned polygon.

Lemma 4.5.4 *Among the intersection points that do not lie on the boundary of the data universe, any intersection point V that has a counter less than $k-2$ does not lie on the boundary of the unpruned polygon. Secondly, any intersection point V that has a counter equal to $k-2$ lies on the boundary of the unpruned polygon.*

Proof Consider the vertex V as shown in Fig. 4.11(a) and assume that it has a counter equal to n. The counter of any point u that lies infinitely close to V and lies in the white area is $n+2$. This is because it is pruned by the n bisectors that prune V and the bisectors $B_{a:q}$ and $B_{c:q}$. Note that any point u that is infinitely close to V can be pruned by at most $n+2$ bisectors (n bisectors that prune V and $B_{a:q}$ and $B_{c:q}$). If the counter of V is less than $k-2$ then the counter of any such point u is always smaller than k. Hence, u is a point inside the unpruned polygon. Since every u that lies infinitely close to V (in any direction) is a point of the unpruned polygon, V does not lie on the boundary of the unpruned polygon.

Now, we prove the second part of the lemma. Assume that the counter of V is equal to $k-2$. Clearly, the counter of u is k. Hence, u lies outside the unpruned polygon. Since u is infinitely close to V, V is a point on the boundary of the unpruned polygon.

∎

Lemma 4.5.4 along with Lemma 4.5.2 show that the boundary of the unpruned polygon consists of only the valid intersection points that either lie on the boundary of the data universe or have counters equal to $k-1$ or $k-2$. Hence, the list containing all intersection points sorted according to θ_p is scanned and the points that do not lie on the boundary of the polygon are ignored. Remaining points are connected in sorted order of θ_p to obtain the unpruned polygon. For example, in Fig. 4.11(b), the unpruned polygon is obtained by connecting the vertices in counter clock wise order, i.e., v_{10}, v_2, v_9, v_8, v_7, v_5, v_4 and v_3 in this order.

Complexity of line 1: compute new intersection points and their counters

The number of new intersection points is $O(m)$ because each existing bisector intersects the new bisector $B_{f:q}$ at most once. To compute the counter of a new intersection point p, we count the number of existing bisectors that prune p. Hence, computing the counter of a new intersection point takes $O(m)$. Since there are $O(m)$ new intersection points, the complexity of computing the counters of these points is $O(m^2)$. Next, we show that the complexity can be reduced to $O(km)$.

Let p be an intersection point between $B_{f:q}$ and an existing bisector. If p lies outside the current influence zone (the unpruned polygon) then its counter is at least equal to k and p can be discarded for this reason. Hence, the counters of only the intersection points that lie inside the unpruned polygon are to be computed. We implement the whole procedure in two steps: 1) for each intersection point p, check whether p lies inside the unpruned polygon or not; 2) for each intersection point p that lies inside the unpruned polygon, compute its counter.

First we show that the step 1 can be implemented in $O(km)$ by using even-odd test [Hai94] to determine if p lies inside the unpruned polygon or not. According to an even-odd test, a point p lies inside a polygon if and only if, for any ray starting from the point p, there is an odd number of crossings of this ray with the edges of the polygon. For example, in Fig. 4.12, point p lies outside the polygon $ABCDEFG$ and any ray starting from p intersects the edges of the polygon even number of times. For instance, the ray starting from p in the direction of x intersects the polygon at two points (w and x). Hence, p lies outside the polygon. Now, we show that for any intersection point p, we can conduct even-odd test in $O(k)$. Since we have at most $O(m)$ new intersection points, this ensures the overall complexity of $O(km)$.

Assume that p is an intersection point of $B_{f:q}$ and an existing bisector B_1 as shown in Fig. 4.12. Note that B_1 is an existing bisector and the algorithm maintains the existing valid intersection points of B_1. For example, the system maintains the intersection points u, v, w and x. To determine the number of intersections of the ray starting from p with the boundary of the unpruned polygon, we simply count the number of valid intersection points of B_1 that lie on the right side of p and lie on the boundary of the unpruned polygon. In Fig. 4.12, such intersection points are w and x. Since B_1 has at most $2k$ intersection points (Lemma 4.5.1), determining the intersection points that lie on the right side of p and lie on the boundary of the unpruned polygon (by using Lemma 4.5.4) takes $O(k)$ if we use a linear scan on all the intersection points related to B_1.

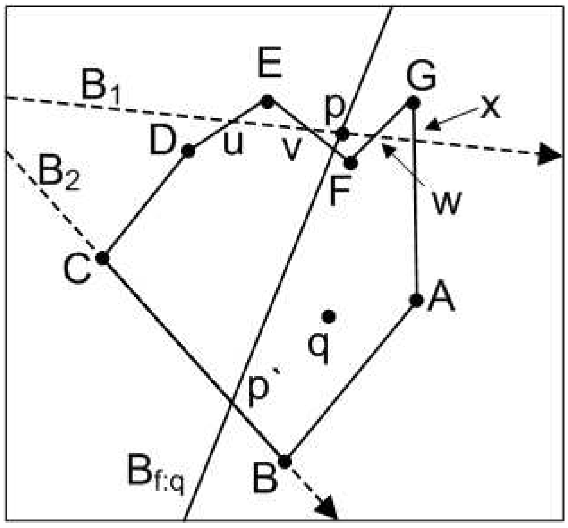


Figure 4.12: Even-odd test

As a special case, if the intersection point lies on the boundary of the unpruned polygon we assume as if it lies inside the unpruned polygon. In Fig. 4.12, the intersection point p' between $B_{f:q}$ and B_2 lies on the boundary of the unpruned polygon. Note that any bisector B_2 can contribute at most $O(k)$ edges to the unpruned polygon (a direct implication of Lemma 4.5.1). Hence, to check whether p' lies on an edge of the polygon, we check if it intersects with any edge of the polygon contributed by B_2. It takes $O(k)$.

Now, we show that step 2 can be done in $O(km)$. As inferred from Lemma 4.5.1, the number of intersection points that lie in the unpruned polygon is at most $O(k)$. Computing the counter of one intersection point takes $O(m)$. Hence, the total complexity of step 2 is $O(km)$.

Recall that line 4 of Algorithm 6 requires the list of intersection points to be sorted in order of θ_p. Hence, we insert each new intersection point in the list of existing intersection points in sorted order. Since the number of existing intersection points is bounded by m^2, each insertion takes $O(log\ m)$. As we insert at most $O(k)$ new intersections, the complexity of keeping the list sorted is $O(k \cdot log\ m)$.

Complexity of line 3: discard intersection points with counters at least equal to k

We scan the list of intersection points and remove any intersection point that has a counter at least equal to k. Clearly, the complexity is $O(km)$.

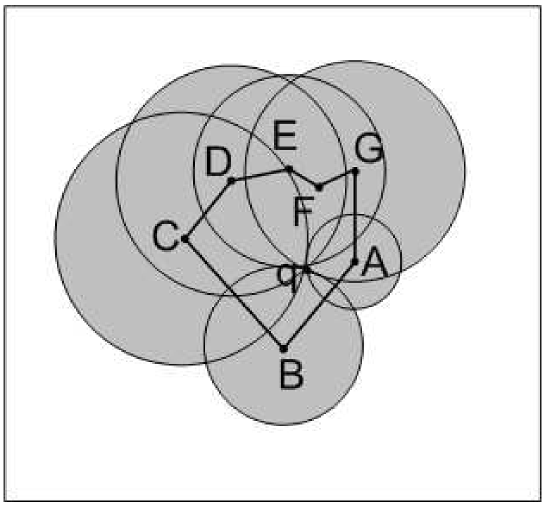
4.6 Handling data updates

In this section, we present techniques to efficiently update the influence zone when the set of facilities changes, i.e., new facilities are added and/or old facilities are deleted. The data updates may be common in many real world applications. For example, consider the example of a restaurant that sends promotional SMS to the people in its influence zone. Note that its influence zone may change when one or more restaurants close or open due to different business timings. Below, we present techniques to efficiently update the influence zone with the change in the set of facilities.

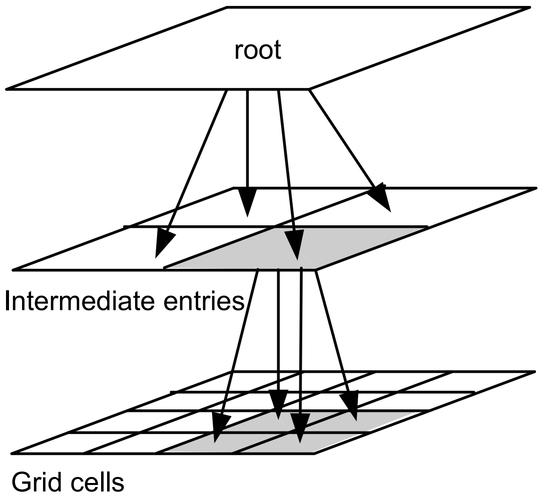
4.6.1 Solution overview

Assume a set of facilities and a set of queries where each query may have a different value of k. Note that a single update (insertion or deletion) may or may not affect the influence zone of a particular query q. Hence, it is important to identify the queries that are affected by an update. To enable us to quickly identify the affected queries, we define *impact region* of a query. The impact region of a query q is the area covered by C_v for every convex vertex v of the influence zone of q. Fig. 4.13(a) shows an example where the influence zone of q is the polygon $ABCDEFG$ and the impact region is shown shaded.

As inferred from Lemmas 4.3.7 and 4.3.8, a facility f affects the influence zone of a query q if and only if f lies in C_v for at least one convex vertex v of the influence zone. Hence, a query is affected by a facility if and only if f lies in the impact region of the query. To quickly identify the queries that are affected by an update, we index the impact regions of all the queries by a grid data structure. Each cell c of the grid has a list



(a) Impact region



(b) Conceptual grid tree

Figure 4.13: Finding the queries affected by an update

called *qList*. The *qList* of a cell c contains every query q such that the impact region of q overlaps or contains the cell c. This list helps in identifying the queries that are affected by an update of a facility in cell c. Along with each query q stored in the qList of cell c, we associate a vertex list called $q.vList$ which consists of every vertex v of the influence zone of q such that C_v overlaps or contains the cell c. This list is to quickly identify the vertices of the influence zone that may contain a given facility in its circle.

When the influence zone of a query q is computed, we add q in every cell c of the grid that overlaps or is contained by the impact region of q. To efficiently do this, we use *conceptual grid tree* which we introduced in Chapter 3.

Fig. 4.13(b) shows an example of the conceptual grid-tree of a 4×4 grid. For a grid-based structure containing $2^n \times 2^n$ cells where $n \geq 0$, the root of our conceptual grid-tree is a rectangle that contains all $2^n \times 2^n$ cells. Each entry at l-th level of this grid-tree contains $2^{(n-l)} \times 2^{(n-l)}$ cells (root being at level 0). An entry at l-th level is divided into four equal non-overlapping rectangles such that each such rectangle contains $2^{(n-l-1)} \times 2^{(n-l-1)}$ cells. Any n-th level entry of the tree corresponds to one cell of the grid structure. Fig. 4.13(b) shows root entry, intermediate entries and the cells of grid. Note that the grid-tree does not exist physically, it is just a conceptual visualisation of the grid.

We identify the cells that overlap with or are contained by the impact region in a hierarchical way by using the grid-tree. For example, if an entry completely lies in the impact region, q is added in the *qList* of all the cells contained in this entry. For more details, please see [HCQL10] or [HCLZ11].

Assume that a facility f is inserted or deleted. We use the location of f to find the cell c of grid relevant to its location. Then, we mark each query q in the qList of c as affected if f is contained by the impact region of q. For each of the affected queries, we update the influence zone accordingly. First, we show how to update the influence zone of a query q if the affecting facility f is a newly added facility. Then, we show the procedure to update the influence zone of q when a facility f is deleted.

4.6.2 Handling an insertion

As stated earlier, we use qList to identify every query that contains f in its impact region. We update the influence zone of each of such query by calling Algorithm 6. As shown earlier, the complexity of Algorithm 6 is $O(km)$. Next, we present few geometric observations that although do not reduce the complexity but help to give more insight into the properties of the problem.

Recall that, at line 1 of Algorithm 6, we compute new intersection points between $B_{f:q}$ and all existing bisectors and then compute their counters. Next, we present few geometric observations that show that we do not need to consider the intersection points of the new bisector $B_{f:q}$ with *all* of the existing bisectors.

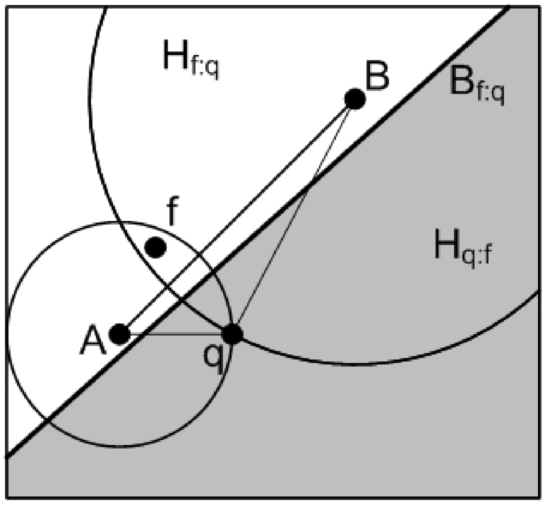
Lemma 4.6.1 *Given a line segment AB and a facility f, the bisector $B_{f:q}$ intersects the line segment AB if and only if exactly one of C_A or C_B contains f, i.e., if both of C_A and C_B contain f or none of C_A and C_B contain f then $B_{f:q}$ does not intersect AB.*

Proof First, we show that $B_{f:q}$ intersects AB only if exactly one of C_A or C_B contains f. We prove this by showing that $B_{f:q}$ does not intersect AB if either both of C_A and C_B contain f or none of C_A or C_B contains f.

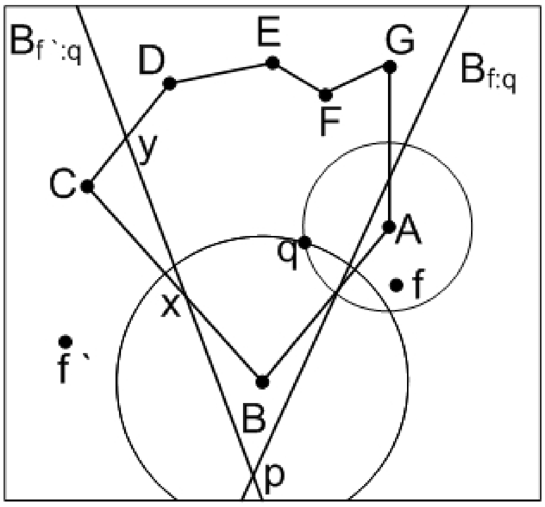
Consider the example of Fig. 4.14(a) where the line segment AB and the circles C_A and C_B are shown. Recall that the bisector $B_{f:q}$ divides the space in two half planes. $H_{f:q}$ denotes the plane that contains f (the white area) and $H_{q:f}$ denotes the plane that contains q (the shaded area). If both C_A and C_B contain f then it means that the bisector $B_{f:q}$ prunes both A and B, i.e., both A and B lie in $H_{f:q}$ (as shown in Fig. 4.14(a)). Since $B_{f:q}$ is a line, the whole line segment AB lies in the plane $H_{f:q}$ which implies that $B_{f:q}$ does not intersect AB.

If none of C_A or C_B contains f then the bisector $B_{f:q}$ does not prune any of A or B. In other words, both A and B lie in $H_{q:f}$. Since $B_{f:q}$ is a line, the whole line segment AB lies in the plane $H_{q:f}$. This implies that $B_{f:q}$ does not intersect AB.

Now, we show that $B_{f:q}$ intersects AB if exactly one of C_A or C_B contains f. Without loss of generality, assume that C_A contains f and C_B does not contain f. This means that the bisector $B_{f:q}$ prunes A and does not prune B. In other words, A lies in $H_{f:q}$ and B lies in $H_{q:f}$. Since $B_{f:q}$ is a line, the line segment AB intersects $B_{f:q}$. ∎



(a) Lemma 4.6.1



(b) Lemma 4.6.2

Figure 4.14: Lemmas 4.6.1 and 4.6.2

The above lemma shows that we may not need to compute the intersection of $B_{f:q}$ with all of the existing bisectors. Let A and B be two end points of a bisector within the influence zone. We only need to compute the intersection of $B_{f:q}$ with the bisector if exactly one of C_A or C_B contains f. However, we first need to efficiently identify such bisectors. Before we show how to identify such bisectors, we define few terms and notations.

Let v be a vertex such that C_v contains f. We call such a vertex v a container vertex. In Fig. 4.14(b), A is a container vertex because C_A contains f. Any edge XY of the influence zone is called a container edge if *at least* one of C_X or C_Y contains f. Any edge that is not a container edge is called a non-container edge. In Fig. 4.14(b), AB and AG are the only container edges. The next lemma shows that we only need to consider the intersection of $B_{f:q}$ with the existing bisectors that intersect with a container edge.

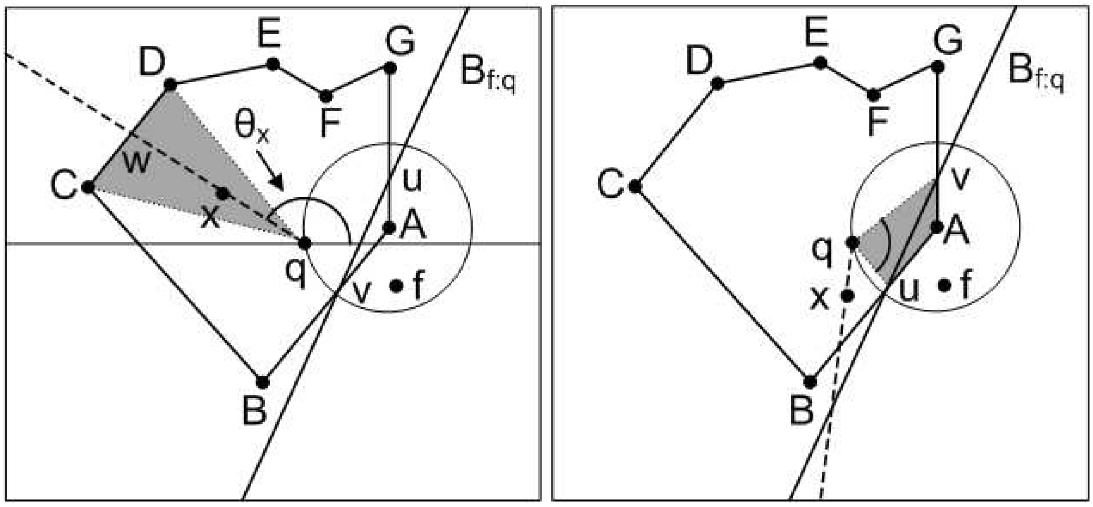
Lemma 4.6.2 *Let $B_{f':q}$ be a bisector that does not intersect with any of the container edges of the influence zone. The intersection point of $B_{f':q}$ and $B_{f:q}$ lies outside the influence zone, i.e., the intersection has a counter at least equal to k and can be ignored for this reason.*

Proof Consider the example of Fig. 4.14(b) where a polygon $ABCDEFG$ is shown. A is the only container vertex of the polygon. The bisector $B_{f':q}$ does not intersect any of the container edges AB or AG. Without loss of generality, assume that the two end points of the bisector $B_{f':q}$ that lie within the influence zone are x and y (see Fig. 4.14(b)). We prove the lemma by showing that the bisector $B_{f:q}$ does not intersect the line segment xy. We show that both C_x and C_y do not contain f which implies (see Lemma 4.6.1) that $B_{f:q}$ does not intersect xy.

We prove that C_x does not contain f and the proof for C_y is similar. As inferred by Lemma 4.3.4, C_x is contained by $C_B \cup C_C$. Since BC is a non-container edge, both C_B and C_C do not contain f. This implies that C_x does not contain f because $C_x \subseteq C_B \cup C_C$. ∎

As inferred from Lemma 4.6.2, we only need to check the intersection of $B_{f:q}$ with the bisectors that intersect with any of the container edges. Next issue is to determine the container edges efficiently. Recall that, in our grid structure, we maintain $q.vList$ for each cell c that contains the list of the vertices that overlap or contain the cell c. If a facility f lies in the cell c, we use $q.vList$ and can identify the vertices of the influence zone of q that contain f. These vertices are the container vertices and the related container edges can be easily determined.

Recall that line 2 of Algorithm 6 requires updating the counters of all existing intersection points. As stated earlier, we increment the counter of an intersection point p if and only if $B_{f:q}$ prunes p. The number of existing intersection points is $O(km)$. Next, we show that we may not need to check whether $B_{f:q}$ prunes p for all of the intersection points.



(a) Lemma 4.6.3 (b) Optimization

Figure 4.15: Optimizations

First, we define few terms and notations. Let x be a point inside the influence zone. *Beam* of x is a line starting from q that passes through the point x. Fig. 4.15(a) shows the beam of a point x in broken line.

Lemma 4.6.3 *An intersection point x is not pruned by a bisector $B_{f:q}$ if the beam of x does not intersect with any container edge of the influence zone.*

Proof Consider the example of Fig. 4.15(a) where A is the only container vertex. Recall that a point x is pruned by a bisector $B_{f:q}$ if and only if C_x contains f. Without loss of generality, assume that the beam of x intersects the influence zone at a non-container edge CD at a point w (see Fig. 4.15(a)). From Lemma 4.3.2, C_x is contained by C_w. From Lemma 4.3.4, C_w is contained by $C_C \cup C_D$. The object f is not contained in $C_C \cup C_D$ because CD is a non-container edge. Hence, f is not contained by C_x which implies that $B_{f:q}$ does not prune x. ∎

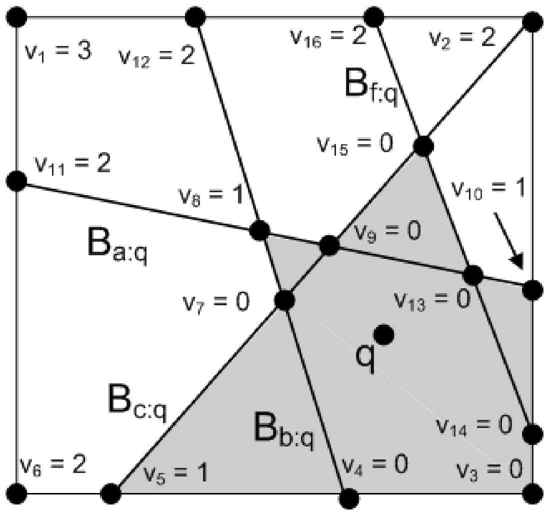
From above lemma, we know that we only need to update the counters of an intersection point if its beam intersects a container edge. Next issue is to efficiently determine the intersection points for which their beams intersect with a container edge. Recall that, for any point x, θ_x is the angle between line qx and the horizontal line passing through q (see Fig. 4.15(a)). For the edge CD in the Fig. 4.15(a), note that the beam of any intersection point x intersects CD if and only if θ_x lies between the angle range θ_D and θ_C (i.e., x lies in the shaded area). Hence, we can use the θ_p of an intersection point p to test if its beam intersects an edge or not.

We further improve the above observation. Consider the example of Fig. 4.15(b), where the intersection point x is shown and its beam intersects a container edge AB. Although the beam of x intersects a container edge, x is not pruned by $B_{f:q}$ as shown in Fig. 4.15(b). Assume that the bisector $B_{f:q}$ intersects the influence zone at two points u and v as shown in Fig. 4.15(b). An intersection point x can be pruned by $B_{f:q}$ only if θ_x is greater than θ_u and is smaller than θ_v (i.e., x lies in the shaded area of Fig. 4.15(b)). The proof is straight forward and is omitted.

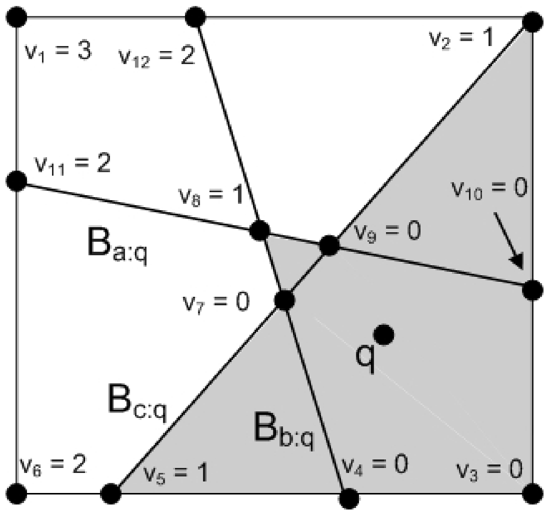
We can quickly identify the intersection points that lie within the shaded area as follows. Recall that we keep the list of intersection points sorted in order of their θ_p. We do a binary search on this list and obtain the first intersection point p that has θ_p just greater than θ_u. Then, the list is scanned in sorted order until the next intersection point p' has $\theta_{p'}$ greater than θ_v. Let n be the number of intersection points that lie in the shaded area of Fig. 4.15(b), the above procedure can find all such intersection points in $O(n + log\ m)$. Hence, the complexity of updating the counters of existing intersection points is $O(n + log\ m)$ where n is at most equal to $O(km)$ (the number of all existing intersection points).

4.6.3 Handling a deletion

If the deleted facility f lies inside the impact region of q then it means that the facility f contributes a bisector to the influence zone. Assume that the influence zone was determined by considering m facilities. When f is deleted, we create the new unpruned polygon P by considering the bisectors of remaining $m-1$ facilities. During the creation of the new unpruned polygon P, we use the following optimizations to improve the efficiency.



(a) Before deleting $B_{f:q}$



(b) After deleting $B_{f:q}$

Figure 4.16: Handling a deletion

1. The counter of any intersection point p that is not pruned by $B_{f:q}$ remains unaffected. Hence, the counters of all such intersection points are not required to be recomputed. This also implies that the part of the influence zone that lies in $H_{q:f}$ remains unaffected.

Consider the example of Fig. 4.16(a) that shows the influence zone ($k = 2$), intersection points and their counters before a facility f and its corresponding bisector $B_{f:q}$ is deleted. The influence zone is shown shaded and it contains the intersection points that have counters less than k. Fig. 4.16(b) shows the new unpruned polygon P, intersection points and their counters after f is deleted. Note that the counters of all the intersection points that are not pruned by $B_{f:q}$ (i.e., the intersection points on the left side of $B_{f:q}$) remain unchanged. Also, the part of the influence zone that lies on the left side of $B_{f:q}$ remains unaffected.

2. The counter of any existing intersection point that is pruned by $B_{f:q}$ is decremented by 1. Hence, the counter of such intersection point is not needed to be computed from scratch. The counter of any new intersection point that is pruned by $B_{f:q}$ is recomputed.

In the example of Fig. 4.16(a), there is only one valid intersection point v_{10} that is pruned by $B_{f:q}$. Its counter is decremented by one after the deletion. Note that the intersection point v_2 had a counter equal to $k = 2$ before f was deleted. Hence, v_2 was not maintained before the deletion of f. The counter of such intersection point is needed to be recomputed.

Note that the new unpruned polygon P is always larger than the previous influence zone. Hence, there may be a facility f' that affects the new unpruned polygon P but was not considered before. To identify all such facilities, we check if there exists a new facility f' that lies in any C_v for any convex vertex of the new unpruned polygon. We can do this by calling Algorithm 5 with two small changes. Firstly, at line 1, the influence zone Z_k is initialized to the new unpruned polygon P instead of initializing it to the whole data universe. Secondly, the algorithm ignores any facility f that had already been considered to construct the influence zone.

Finally, we present another minor optimization. Note that at line 5 of Algorithm 5, we check if an entry e of R-tree is contained by C_v for every convex vertex of the influence zone. However, note that there are some convex vertices of the unpruned polygon P (see Fig. 4.16(b)) that existed in previous influence zone (see Fig. 4.16(a)). For example, the convex vertex v_8 is a convex vertex of the previous influence zone as well as the new unpruned polygon. Hence, we do not need to consider v_8 at line 5 of Algorithm 5. This is because if there was a facility in the circle of such convex vertex, that would have been considered before. Hence, the convex vertices that existed in the influence zone before the deletion can be ignored at line 5 of Algorithm 5.

4.7 Experiments

In Section 4.7.1, we evaluate the performance of our algorithms for snapshot RkNN queries. Since computation of the influence zone is a sub-task of the snapshot RkNN queries, we evaluate the cost of computing influence zone while evaluating the performance of RkNN algorithms. In Section 4.7.2, we evaluate the performance of our algorithm for continuous monitoring of RkNN queries. Finally, in Section 4.7.3, we evaluate our techniques for updating the influence zone when the underlying data set is changed due to insertions and deletions.

4.7.1 Snapshot RkNN queries

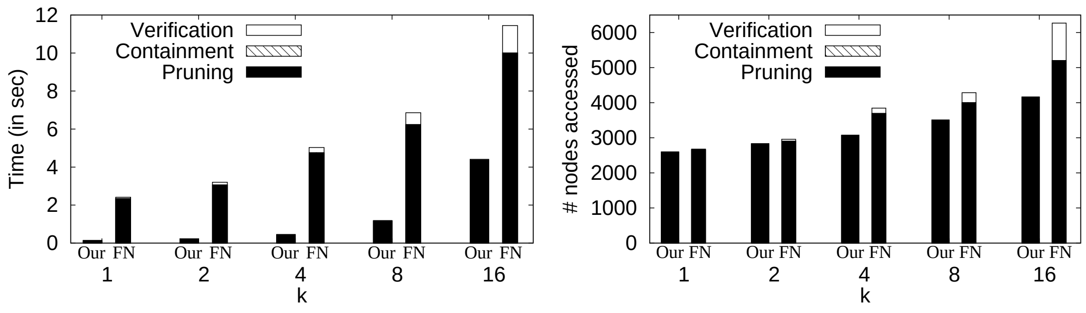
For monochromatic and bichromatic RkNN queries, we compare our algorithm with the best known existing algorithm called FINCH [WYCT08b]. We use both synthetic and real data sets. Each synthetic data set consists of 50000, 100000, 150000 or 200000 points following either Uniform or Normal distribution. The real data set consists of $175,812$ extracted locations in North America[4] and we randomly divide these points into two sets of almost equal sizes. One of the sets corresponds to the set of facilities and the other to the set of users. Following the experiment settings used in [WYCT08b] for FINCH, the page size is set to 4096 bytes and the buffer size is set to 10 pages which uses random eviction strategy. We use the two real data sets to evaluate the performance unless mentioned otherwise. We vary k from 1 to 16 and the default value is 8. From the set of facilities, we randomly choose 500 points as the query points. The experiment results correspond to the total cost of processing these 500 queries.

As stated in Chapter 2, FINCH has three phases namely pruning, containment and verification. Our algorithm has only pruning and containment phases. We show the CPU and IO cost of each phase for both of the algorithms. Experiment results demonstrate that our algorithm outperforms FINCH in terms of both CPU time and the number of nodes accessed. FINCH is denoted as FN in the experiment figures.

[4]http://www.cs.fsu.edu/ lifeifei/SpatialDataset.htm

Monochromatic RkNN queries

In Fig. 4.17, we vary the value of k and study the effect on both of the algorithms. The cost of containment phase is negligible for both of the algorithms. Note that the pruning phase corresponds to the cost of computing the influence zone for our algorithm. The cost of computing the influence zone is even smaller than the pruning cost of FINCH which prunes less area than our algorithm. CPU cost of our algorithm is lower mainly because we use efficient checks to prune the entries of the R-tree and because we do not need to compute the convex hull (in contrast to FINCH that computes a convex polygon to approximate the unpruned area).



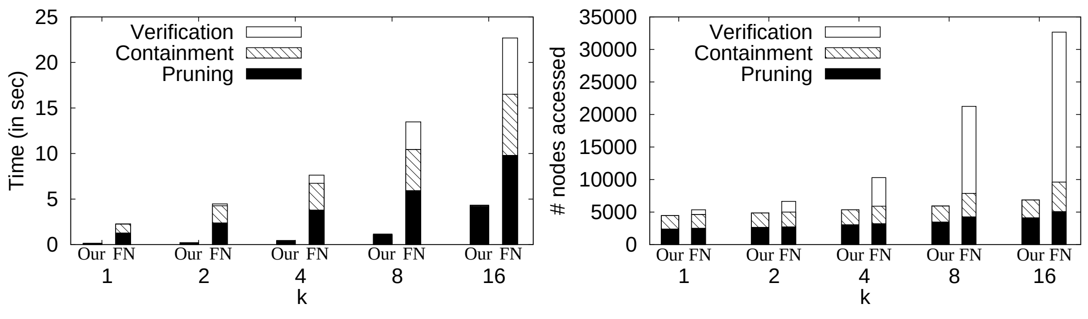
(a) CPU time (b) Nodes accesses

Figure 4.17: Effect of k (monochromatic RkNN)

Although we access more facility points to prune the space, the IO cost of computing the influence zone is slightly lower than the pruning cost of FINCH. This is mainly because these facility points are usually found in 1 or 2 leaf nodes which are accessed by FINCH anyway because they are too close to the query. The unpruned area of our algorithm is smaller as compared to FINCH which results in pruning more nodes of the R-tree.

Bichromatic RkNN queries

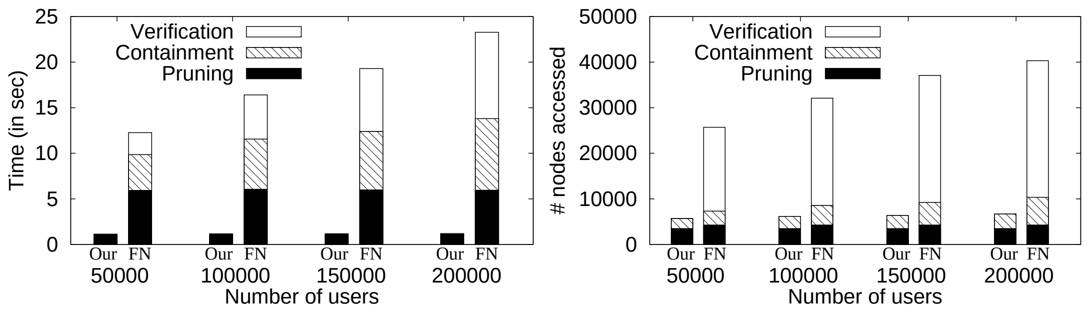
Fig. 4.18 studies the effect of k on the cost of bichromatic RkNN queries. The CPU time taken by containment phase of our algorithm is much smaller as compared to FINCH. This is mainly because i) the unpruned area of our algorithm is smaller and ii) we use efficient



(a) CPU time (b) Nodes accesses

Figure 4.18: Effect of k (bichromatic RkNN)

containment checking to prune the entries and the objects. IO cost of the containment phase is also smaller for our algorithm because the unpruned area of our algorithm is smaller. Our algorithm does not require the verification. On the other hand, FINCH consumes significant amount of CPU time and IOs in the verification phase.

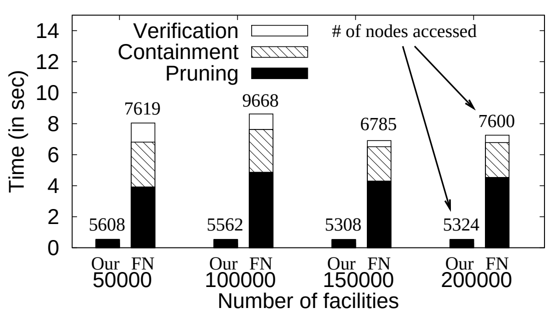


(a) CPU time (b) Nodes accesses

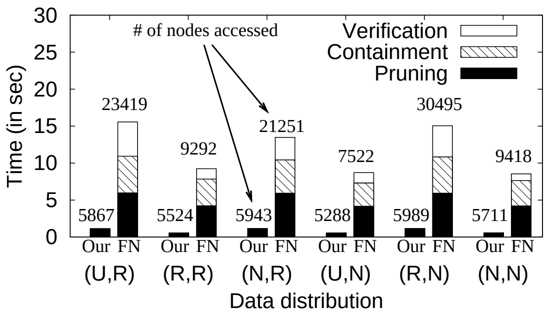
Figure 4.19: Effect of number of users

Fig. 4.19 studies the effect of the number of the users on both of algorithms. The set of facilities corresponds to the real data set and the locations of the users follow normal distribution. Our algorithm scales much better. On the other hand, the cost of FINCH degrades with the increase in the number of users because a larger number of users are within the unpruned area and require verification.

In Fig. 4.20(a), we study the effect of the number of the facilities. The set of the users correspond to the real data set and the locations of the facilities follow normal distribution. Both of the algorithms are not significantly affected by the increase in the number of the facilities and our algorithm performs significantly better than FINCH.



(a) number of facilities



(b) Data distribution

Figure 4.20: Effect of data size and distribution

Fig. 4.20(b) studies the effect of the data distribution on both of the algorithms. The data distributions of the facilities and the users are shown in the form ($Dist_1$,$Dist_2$) where $Dist_1$ and $Dist_2$ correspond to the data distribution of the facilities and the users, respectively. U, R and N correspond to Uniform, Real and Normal distributions, respectively. For example, (U,R) corresponds to the case where the facilities follow uniform distribution and the users correspond to the real data set. Each data set contains around $88,000$ objects. Our algorithm outperforms FINCH both in terms of CPU time and the number of nodes accessed for all of the data distributions.

Fig. 4.21 studies the effect of the buffer size on both of the algorithms. As the pruning and the containment phases do not visit a node twice, our algorithm is not affected by the buffer size. FINCH issues multiple range queries to verify the candidate objects. For this reason, the cost of its verification phase depends on the buffer size. Note that FINCH performs worse than our algorithm even when it uses large buffer size. Number of nodes accessed by FINCH is around $194,000$ and $61,000$ when the buffer size is 2 and 5, respectively.

Verification of theoretical analysis

In Fig. 4.22 and Fig. 4.23, we vary k and verify the theoretical analysis presented in Section 4.5. In all three experiments, we run bichromatic RkNN queries on uniform data sets consisting of $100,000$ facilities and the same number of users.

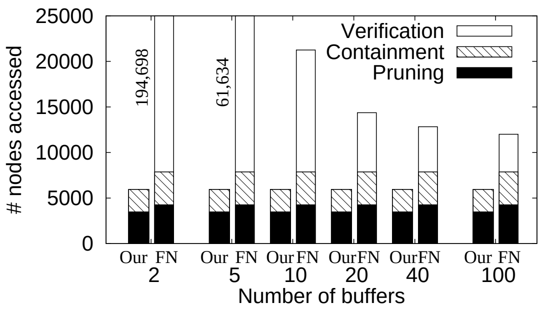


Figure 4.21: Buffer size

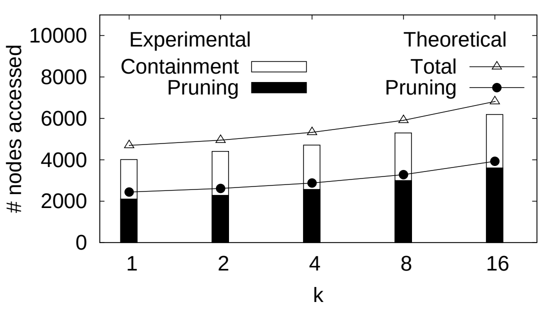
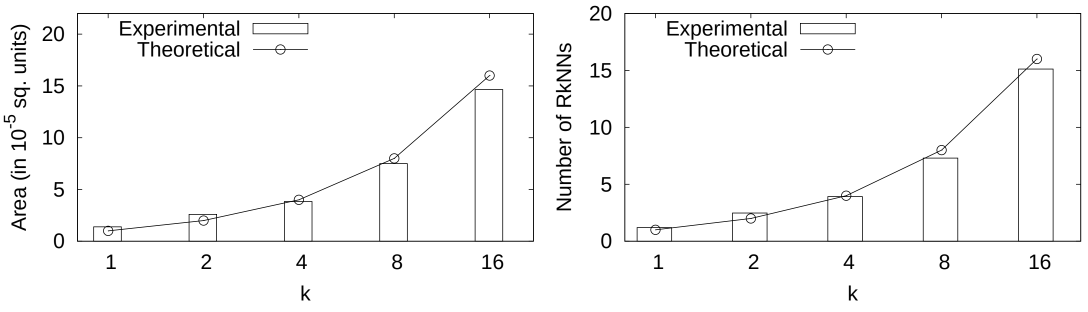


Figure 4.22: Theoretical analysis (IO cost)

In Fig. 4.22, we compare the experimental value of total number of nodes accessed with the theoretical value. Recall that the pruning phase of our algorithm corresponds to the computation of the influence zone. Fig. 4.22 shows the accuracy of our theoretical analysis of the IO cost of computing the influence zone and the total cost of our RkNN algorithm.



(a) Area of influence zone (b) Number of RkNNs

Figure 4.23: Theoretical analysis

In Fig. 4.23(a) and Fig. 4.23(b), we vary k and verify our theoretical analysis of the area of the influence zone and the number of RkNNs, respectively. It can be seen that the theoretical results are close to the experimental results and follow the trend.

4.7.2 Continuous Monitoring of RkNN

As mentioned earlier, the problem addressed by the influence zone based algorithm is a special case of the continuous RkNN queries. Hence, it is not fair to use the existing best known algorithms without making any obvious changes that improve the performance. Lazy Updates [CLZ$^+$09] is the best known algorithm for continuous monitoring of RkNN queries (even for this special case, we observe that it outperforms other algorithms after necessary changes are made to all the existing algorithms). Hence, we compare our algorithm with Lazy Updates.

To conduct a fair evaluation, we set the size of the *safe region* for the Lazy Updates algorithm to zero. This is because the facilities do not move and the safe regions will not be useful in this case. We tested different possible sizes of the safe region and confirmed that this is the best possible setting for Lazy Updates for this special case of the continuous RkNN query.

Table 4.2: System parameters

Parameter	**Range**
Number of users ($\times$1000)	40, 60, 80, **100**, 120
Number of facilities ($\times$1000)	40, 60, 80, **100**, 120
Number of queries	100, 300, **500**, 700, 1000
k	1, 2, 4, **8**, 16
Speed of objects (users) in km/hr	40, 60, **80**, 100, 120
Mobility of objects (users) in %	5, 20, 40, 60, **80**, 100

Our experiment settings are similar to the settings used in [CLZ+09] by Lazy Updates. More specifically, we use Brinkhoff generator [Bri02] to generate the users moving on the road map of Texas (data universe is approximately 1000Km$\times$1000Km). The facilities are randomly generated points in the same data universe. Table 4.2 shows the parameters used in our experiments and the default values are shown in bold.

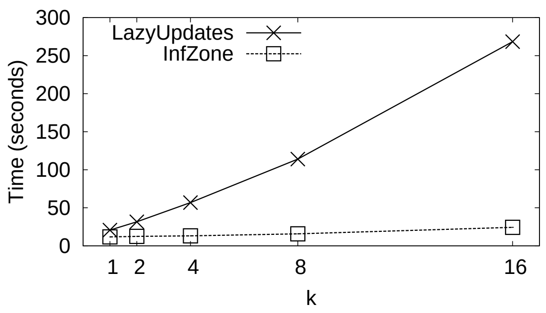
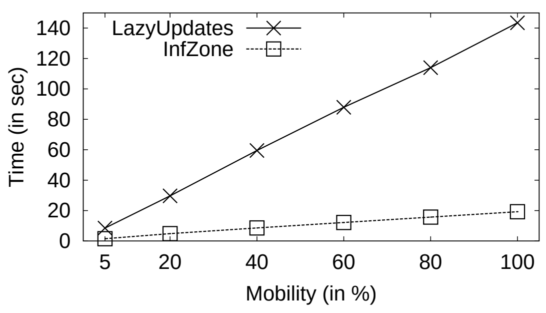
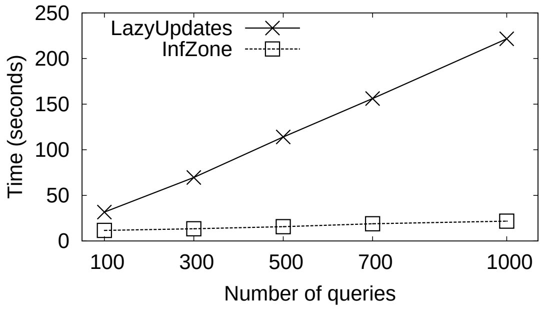


Figure 4.24: Effect of k

The locations of the users are reported to the server after every one second (i.e., timestamp length is one second). The mobility of the objects refers to the percentage of the objects that report location updates at a given timestamp. In accordance with [CLZ+09], the grid cardinality of both of the algorithms is set to 64×64. Each query is monitored for 5 minutes (300 timestamps) and the total time taken by all the queries is reported.

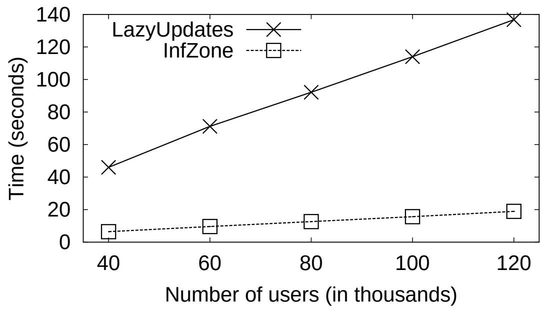


(a) Mobility

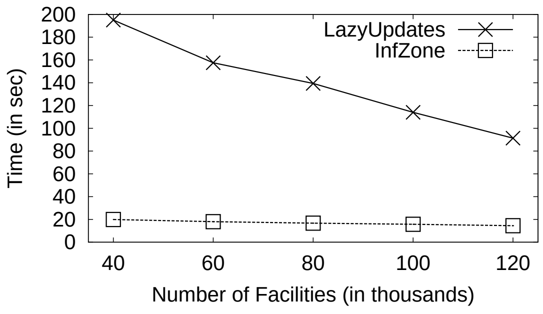


(b) # of queries

Figure 4.25: Effect of mobility and number of queries



(a) # of users



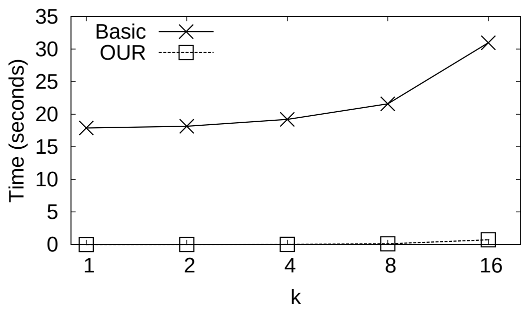
(b) # of facilities

Figure 4.26: Effect of data size

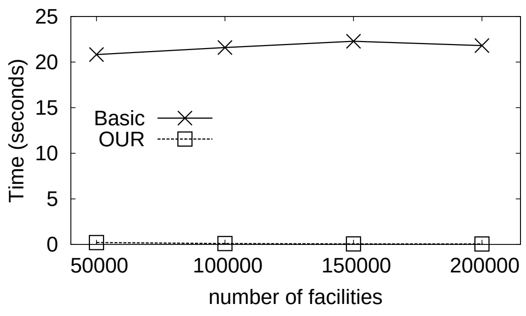
In Fig. 4.24, 4.25(a), 4.25(b), 4.26(a) and 4.26(b), we study the effect of k, the data mobility, the number of the queries, the number of the users and the number of the facilities, respectively. Influence zone based algorithm is shown as *InfZone*. Clearly, the influence zone based algorithm outperforms Lazy Updates for all the settings and scales better. In Fig. 4.26(b), both of the algorithms perform better as the number of facilities increases. This is because the unpruned area becomes smaller when the number of facilities is large. Hence, a smaller area is to be monitored by both the algorithms and it results in lower cost.

4.7.3 Handling data updates

We compare our proposed technique with BASIC algorithm. BASIC calls Algorithm 6 whenever a new facility is added and recomputes the influence zone from scratch whenever a facility that contributes to the existing influence zone is deleted. We randomly generate 1000 updates such that half of the updates are insertions and the other half consists of deletions. The default value of k is 8, number of facilities in the default data set is $100,000$ and the number of queries is 500. The influence zone of each of the query is updated after every data update and the results show the total cost of handling all data updates.



(a) effect of k



(b) # of facilities

Figure 4.27: Handling data updates

Fig. 4.27 compares our approach with BASIC approach for increasing k and increasing number of facilities. Fig. 4.27(a) shows that our proposed technique not only performs significantly better than BASIC approach but also scales better as the value of k increases. Fig. 4.27(b) shows that both of the algorithms are not significantly affected as the number of facilities increases.

4.8 Summary

In this chapter, we introduced the concept of influence zone which is the area such that every point inside this area is the RkNN of q and every point outside this area is not the RkNN. The influence zone has several applications in location based services, marketing and decision support systems. We show that it can also be used to efficiently process

RkNN queries. First, we present efficient algorithm to compute the influence zone. Then, based on the influence zone, we present efficient algorithms to process RkNN queries that significantly outperform existing best known techniques for both the *snapshot* and *continuous* RkNN queries. We also present a detailed theoretical analysis to analyse the area of the influence zone and IO costs of our RkNN processing algorithms. Our experiments demonstrate the accuracy of our theoretical analysis. We also conduct a rigorous complexity analysis and show that the complexity of one of our proposed algorithms can be reduced from $O(m^2)$ to $O(km)$ where $m > k$ is the number of objects used to compute the influence zone. We show that our techniques can be applied to dimensionality higher than two and we present efficient techniques to handle data updates.

Chapter 5

Reverse Nearest Neighbors Queries on Uncertain Data

In this chapter, we formalize and study probabilistic RNN query that is to find the probable reverse nearest neighbors on uncertain data with probability higher than a given threshold. This research was published in [CLW+10].

5.1 Overview

Uncertain data is inherent in many important applications such as sensor databases, moving object databases, market analysis, and quantitative economic research. In these applications, the exact values of data might be unknown due to limitation of measuring equipment, delayed data updates, incompleteness, or data anonymization to preserve privacy.

Usually an uncertain object is represented in two ways: 1) using a probability density function [BSI08, CCMC08] (continuous case) and 2) using all possible instances [Wid05, PJLY07] each with an assigned probability (discrete case). Our focus in this chapter is to investigate the discrete cases.

Probabilistic RNN queries have many applications. Consider the example in Fig. 5.1, where three residential blocks A, B and Q are shown. The houses within each block are shown as small circles. The centroid of each residential block is shown as a hollow triangle. For privacy reasons, we may only know the residential blocks in which the people live (or zip code) but we do not have any information about the exact addresses of their houses. We can assign some probability to each possible location of a person in his residential block. e.g; the exact location of a person living in A is a_1 with 0.5 probability.

Conventional queries on these residential blocks may use distance functions like the distance between the centroids of two blocks. However, the results provided by the conventional queries may not be meaningful. There are two major limitations for conventional queries on such data[1].

1) The conventional queries do not consider the locations of houses within each residential block. This affects quality of the reported results. For instance, if the distance between centroids of two residential blocks is used as distance function, the closest block of A is B (in other words the person living in A is *not* the RNN of someone living in Q). However, if the locations of houses within each block are considered, we find that for most of the houses in A, the houses in Q are closer than the houses in B. For example, the distance of a_1 to every house in Q is less that its distance to any house in B. Similarly, the distance of a_2 to every house in Q is less than its distance to b_1. Which means, a person living in A has high chances to be the RNN of some person living in Q.

2) Conventional queries do not report the probability of objects to be the answer (an object is either a RNN or not a RNN). On the other hand, probabilistic reverse nearest neighbor queries provide more information by including the probability of an object to be the answer. For example, a probabilistic reverse nearest neighbor query reports that the probability of a person living in block A to become the RNN of a person living in Q is 0.75 according to the possible world semantics (see example 5.2.1). This type of results are more meaningful and interesting.

[1]Other distance functions like maximum distance, minimum distance and aggregated distance also have these limitations.

Probabilistic RNN queries have applications in privacy preserving location-based services where the exact location of every user is obfuscated into a cloaked spatial region [MCA06]. However, the users might still be interested in finding their reverse nearest neighbors. We can model this problem to finding probabilistic reverse nearest neighbor by assigning confidence level to some possible locations of every user within his/her respective cloaked spatial region. Probabilistic RNN queries may also be useful to identify similar trading trends in stock markets. Each stock has many deals. A deal (transaction) is recorded by the price (per share) and the volume (number of shares). For a given stock s, clients may be interested in finding all other stocks that have trading trends more similar to s than others. In such application, we can treat each stock as an uncertain object and its deals as its uncertain instances. There are a number of other applications for the queries that consider the proximity of uncertain objects [BSI08, CCMC08, KKR07] and the applications of RNNs on uncertain objects are very similar.

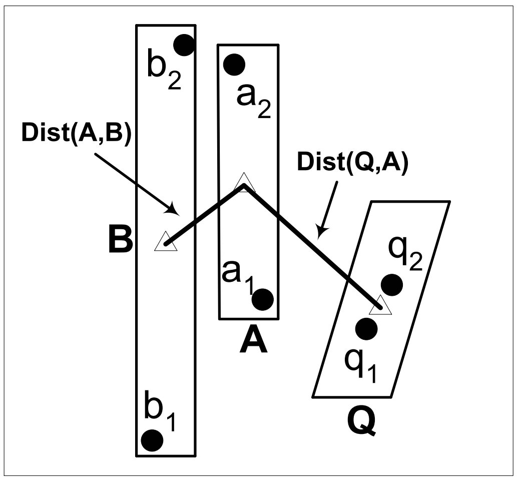


Figure 5.1: An example of a probabilistic RNN query

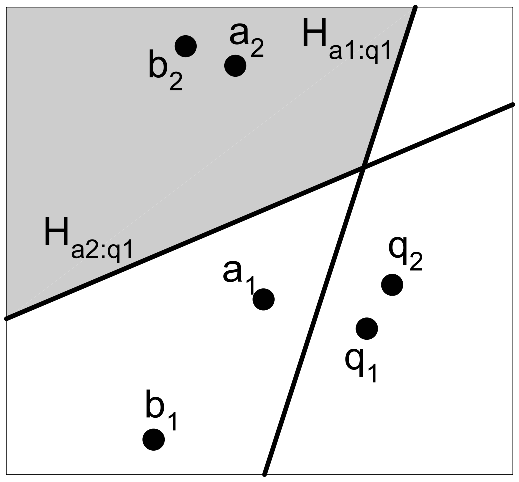


Figure 5.2: Any point in shaded area cannot be RNN of q

Probabilistic RNN query processing poses new challenges in designing new efficient algorithms. Although RNN query processing has been extensively studied based on various pruning methods, these pruning techniques either cannot be directly applied to probabilistic RNN queries or become inefficient. For example, the perpendicular bisectors adopted in the state-of-the-art RNN query processing algorithm [TPL04] assume that objects are spatial points. In contrast, uncertain objects have arbitrary shapes of their uncertain regions. In addition, applying the pruning rules on the instance level of uncertain objects is extremely expensive as each uncertain object usually has a large number of instances.

Another unique challenge in probabilistic RNN queries is that the verification of candidate objects usually incurs substantial cost due to large number of instances in each uncertain object. By verification, we mean computing the exact probability of an object being the RNN of the query and testing whether it qualifies the probabilistic threshold or not. Note that instances from objects that are close to the candidate objects also need to be considered in the verification phase.

We formalize the problem of probabilistic RNN queries on uncertain data using the semantics of *possible worlds*. We present a new probabilistic RNN query processing framework that employs i) several novel pruning approaches exploiting the probability threshold and geometric, topological and metric properties. ii) a highly optimized verification method that is based on careful upper and lower bounding of the RNN probability of candidate objects.

Our contributions are as follows:

- To the best of our knowledge, we are the first to formalize the problem of probabilistic reverse nearest neighbors based on the possible worlds semantics.

- We develop efficient query processing algorithm of probabilistic RNN queries. The new method is based on non-trivial pruning rules especially designed for uncertain data and the probability threshold. Although we focus on *discrete* case where each object is represented by some possible probable instances, our pruning rules can be applied to the *continuous* case where each uncertain object is represented by a probability density function.

- To better understand performance of our proposed approach, we devise a baseline exact algorithm and a sampling-based approximate algorithm. Experiment results on synthetic and real data sets show that our algorithm is much more efficient than the baseline algorithm and performs better than the approximate algorithm for most of the cases and is scalable.

This chapter is organized as follows. In Section 5.2, we formalize the problem and present the preliminaries and notations used in this chapter. Our proposed pruning rules are presented in Section 5.3. Section 5.4 presents our proposed algorithm for answering probabilistic reverse nearest neighbor queries. Section 5.5 evaluates the proposed methods with extensive experiments and Section 5.6 summarizes this chapter.

5.2 Problem Definition and Preliminaries

5.2.1 Problem Definition

Given a set of data points P and a query point q, a conventional reverse nearest neighbor query is to find every point $p \in P$ such that $dist(p,q) \leq dist(p,p')$ for every $p' \in (P-p)$.

Now we define probabilistic reverse nearest neighbor queries. Consider a set of *uncertain objects* $\mathcal{U} = \{U_1, ..., U_n\}$. Each uncertain object U_i consists of a set of *instances* $\{u_1, ..., u_m\}$. Each instance u_j is associated with a probability p_{u_j} called *appearance probability* with the constraint that $\sum_{j=1}^{m} p_{u_j} = 1$. We assume that the probability of each instance is independent of other instances. A *possible world* $W = \{u_1, ..., u_n\}$ is a set of instances with one instance from each uncertain object. The probability of W to appear is $P(W) = \prod_{i=1}^{n} p_{u_i}$. Let Ω be the set of all possible worlds, then $\sum_{W\in\Omega} P(W) = 1$.

The probability $RNN_Q(U_i)$ of any uncertain object U_i to be the RNN of an uncertain object Q in all possible worlds can be computed as;

$$RNN_Q(U_i) = \sum_{(u,q),u\in U_i,q\in Q} p_q \cdot p_u \cdot RNN_q(u) \tag{5.1}$$

$RNN_q(u)$ is the probability that an instance $u \in U_i$ is the RNN of an instance $q \in Q$ in any possible world W given that both u and q appear in W.

$$RNN_q(u) = \prod_{V \in (\mathcal{U} - U_i - Q)} (1 - \sum_{v \in V, dist(u,v) < dist(u,q)} p_v) \tag{5.2}$$

Given a set of uncertain objects $\mathcal{U}$ and a probability threshold ρ, problem of finding probabilistic reverse nearest neighbors of any uncertain object Q is to find every uncertain object $U_i \in \mathcal{U}$ such that $RNN_Q(U_i) \geq \rho$.

Example 5.2.1 *Consider the example of Fig. 5.1 where the uncertain objects A, B and Q are shown. Assume that the appearance probability of each instance is* 0.5*. According to Equation* (5.2)*, $RNN_{q_1}(a_1) = 1$ because a_1 is closer to q_1 than it is to b_1 or b_2. Also $RNN_{q_1}(a_2) = 1 - 0.5$ because $dist(a_2, b_2) < dist(a_2, q_1)$. Note that b_1 does not affect the probability of a_2 to be the RNN of q_1 because $dist(a_2, b_1) > dist(a_2, q_1)$. Similarly, $RNN_{q_2}(a_1) = 1$ and $RNN_{q_2}(a_2) = 0.5$. According to Equation* (5.1)*, $RNN_Q(A) = (0.5 \times 0.5 \times 1) + (0.5 \times 0.5 \times 1) + (0.5 \times 0.5 \times 0.5) + (0.5 \times 0.5 \times 0.5) = 0.75$. RNN probability of B can be computed similarly and $RNN_Q(B) = 0.25$. If the probability threshold ρ is* 0.7*, then the object A is reported as result.*

5.2.2 Preliminaries

The filter-and-refine paradigm is widely adopted in processing RNN queries in spatial databases. The idea is to quickly prune away points which are closer to another point (usually called *filtering point*) than to the query point. The state-of-the-art pruning rule is based on perpendicular bisector [TPL04]. It consists of two phases: the pruning phase and the verification phase.

Hence, some objects are used to filter other objects and are called *filtering objects*. Objects that cannot be filtered are called *candidate objects*. The pruning in RNN query processing involves three objects, the query, the filtering object and a candidate object. We use R_Q, R_{fil} and R_{cnd} to denote the smallest hyper-rectangles enclosing uncertain query object, filtering object and candidate object, respectively.

Table 5.1 defines the symbols and notations used throughout this chapter.

Table 5.1: Notations

Notation	Definition
U	an uncertain object
u_i	i^{th} instance of uncertain object U
$B_{x:q}$	a perpendicular bisector between point x and q
$H_{x:q}$	a half space defined by $B_{x:q}$ containing point x
$H_{q:x}$	a half space defined by $B_{x:q}$ containing point q
$H_{a:b} \cap H_{c:d}$	intersection of the two half spaces
$P[i]$	value of point P in the i^{th} dimension
R_U	minimum bounding rectangle (MBR) enclosing all instances of an uncertain object U

5.3 Pruning Rules

Although the pruning for RNN query processing in spatial databases has been well studied, it is *non-trivial* to devise pruning strategies for RNN query processing on uncertain data. For example, if we naïvely use every instance of a filtering object to perform bisector pruning [TPL04], it will incur a huge computation cost due to large number of instances in each uncertain object. Instead, we devise non-trivial generalization of bisector pruning for minimum bounding rectangles (MBRs) of uncertain objects based on a novel notion of *normalized half space*.

Verification is extremely expensive in probabilistic RNN query processing because, in order to verify an object as probabilistic RNN, we need to take into consideration not only the instances of this object but also the instances of query object and other nearby objects. Hence it is important to devise efficient pruning rules to reduce the number of objects that need verification. In this section, we present several pruning rules from the following orthogonal perspectives:

- Half space based pruning that exploits geometrical properties (Section 5.3.1)
- Dominance based pruning that exploits topological properties (Section 5.3.2)
- Metric based pruning (Section 5.3.3)
- Probabilistic pruning that exploits the probability threshold (Section 5.3.4)

We remark that the first three of the above pruning techniques are the same as those presented in Chapter 3. The only difference is that, in this chapter, we present generalized versions of these pruning rules that can be applied on multidimensional space (and not only on 2d space).

5.3.1 Half Space Pruning

Consider a query point q and a filtering object U that has n instances $\{u_1, u_2, \ldots, u_n\}$. Let $H_{u_i:q}$ be the half space between q and u_i. Any instance $u \notin U$ that lies in $\cap_{i=1}^{n} H_{u_i:q}$ has zero probability to be the RNN of q because by the property of $H_{u_i:q}$, u is closer to every u_i than to q.

Example 5.3.1 *Consider the example of Fig. 5.2 where the bisectors between q_1 and the instances of A are drawn and the half spaces $H_{a_1:q_1}$ and $H_{a_2:q_1}$ are shown. Intersection of the two half spaces is shown shaded and any point that lies in the shaded area is closer to both a_1 and a_2 than q_1. For this reason, b_2 cannot be the RNN of q_1 in any possible world.*

This pruning is very expensive because we need to compute intersection of all half spaces $H_{u_i:q}$ for every $u_i \in U$. Below we present our pruning rules that utilize the MBR of the entire filtering object, R_{fil}, to prune the candidate object with respect to a query instance q or the MBR of uncertain query object Q.

Pruning using R_{fil} and an instance q

First we present the intuition. Consider the example of Fig. 5.3 where we know that the point p lies on a line MN but we do not know the exact location of p on this line. The bisectors between q and the end points of the line (M and N) can be used to prune the area safely. In other words, any point that lies in the intersection of half spaces $H_{M:q}$ and $H_{N:q}$ (grey area) can never be the RNN of q. It can be proved that whatever be the location of point p on the line MN, the half space $H_{p:q}$ always contains $H_{M:q} \cap H_{N:q}$.

Hence any point p' that lies in $H_{M:q} \cap H_{N:q}$ would always be closer to p than to q and for this reason cannot be the RNN of q.

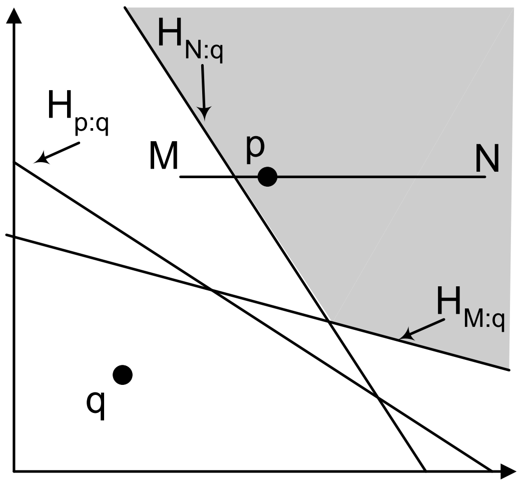


Figure 5.3: The exact location of the point p on line MN is not known

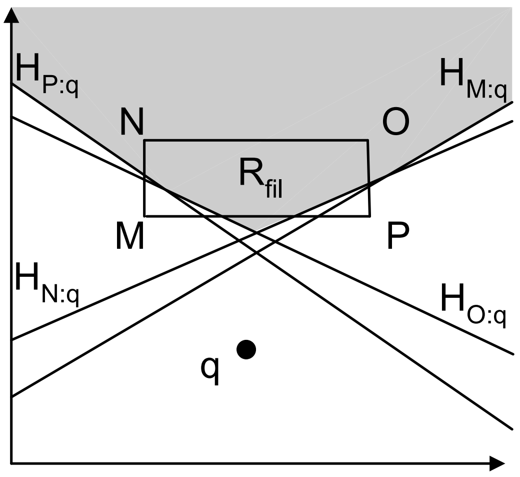


Figure 5.4: Any point in shaded area cannot be RNN of q in any possible world

Based on the above observation, below we present a pruning rule for the case when the exact location of a point p is unknown within some hyper-rectangle R_{fil}.

Pruning Rule 5.3.2 *Let R_{fil} be a hyper-rectangle and q be a query point. For any point p that lies in $\bigcap_{i=1}^{2^d} H_{C_i:q}$ (C_i is the i^{th} corner of R_{fil}), $dist(p,q) > maxdist(p, R_{fil})$ and thus p cannot be the RNN of q.*

Note that this pruning rule is a generalized version of the pruning rule 3.3.4 presented in Chapter 3. The proof is similar to the proof of pruning rule 3.3.4 and is omitted.

Consider the example of Fig. 5.4. Any point that lies in shaded area is closer to every point in rectangle R_{fil} than to q. Note that if R_{fil} is a hyper rectangle that encloses *all* instances of the filtering object U_i then any instance $u \in U_{j,j\neq i}$ that lies in $\bigcap_{i=1}^{2^d} H_{C_i:q}$ can never be the RNN of q in any possible world.

Pruning using R_{fil} and R_Q

Pruning rule 5.3.2 prunes the area such that any point lying in it can never be the RNN of some instance q. However, the points in the pruned area may still be the RNNs of other instances of the query. Now, we present a pruning rule that prunes the area using R_{fil} and R_Q such that any point that lies in the pruned area cannot be the RNN of *any* instance of Q.

Consider the example of Fig. 5.5 where the exact location of the query point q on line MN is not known. Unfortunately, in contrast to the previous case of Fig. 5.3, the bisectors between p and the end points of the line MN do *not* define the area that can be pruned. If we prune the area $H_{p:M} \cap H_{p:N}$ (the grey area), we may miss some point p' that is the RNN of q. Fig. 5.5 shows a point p' that is the RNN of q but lies in the shaded area. This is because the half space $H_{p:q}$ does not contain $H_{p:M} \cap H_{p:N}$. This makes the pruning using R_{fil} and R_Q challenging.

Note that if $H_{p:N}$ is moved such that it passes through the point where $H_{p:q}$ intersects $H_{p:M}$ then $H_{p:M} \cap H_{p:N}$ would be contained by $H_{p:q}$. We note that in the worst case when p lies infinitesimally close to point M, $H_{p:q}$ and $H_{p:M}$ intersect each other at point c which is the centre of line joining p and M. Hence, in order to safely prune the area, the half space $H_{p:N}$ should be moved such that it passes through the point c. The point c is shown in Fig. 5.5. A half space that is moved to the point c is called a *normalized* half space and a half space $H_{p:N}$ that is normalized is denoted as $H'_{p:N}$. Fig. 5.5 shows $H'_{p:N}$ in broken line and $H'_{p:N} \cap H_{p:M}$ (the dotted shaded area) can be safely pruned.

Before we present our pruning rule for the general case that uses 2^d half spaces to prune the area using hyper-rectangles R_Q and R_{fil}, we define the following concepts:

Antipodal Corners: Let C be a corner of rectangle $R1$ and C' be a corner in $R2$, the two corners are called *antipodal corners*[2] if for every dimension i where $C[i] = R1_L[i]$ then $C'[i] = R2_H[i]$ and for every dimension j where $C[j] = R1_H[j]$ then $C'[j] = R2_L[j]$. Fig. 5.6 shows two rectangles $R1$ and $R2$. The corners D and O are antipodal corners. Similarly, other pairs of antipodal corners are (B, M), (C, N) and (A, P).

Antipodal half space: A half space that is defined by the bisector between two antipodal corners is called *antipodal half space*. Fig. 5.6 shows two antipodal half spaces

[2] $R_L[i]$ (resp. $R_H[i]$) is the lowest (resp. highest) coordinate of a hyper-rectangle R in i^{th} dimension

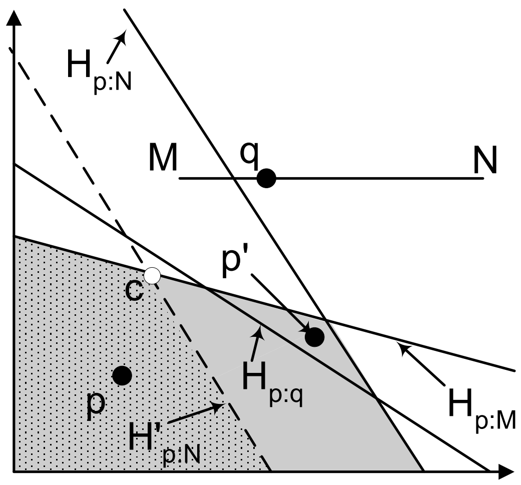


Figure 5.5: Any point in dotted area can never be RNN of q

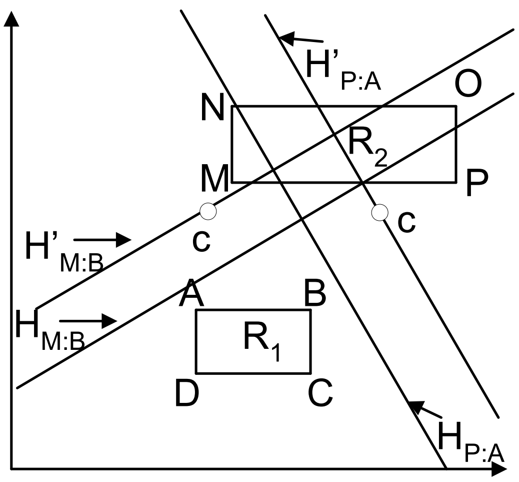


Figure 5.6: Antipodal corners and normalized half spaces

$H_{M:B}$ and $H_{P:A}$.

Normalized half space: Let B and M be two points in hyper-rectangles $R1$ and $R2$, respectively. The normalized half space $H'_{M:B}$ is a space defined by the bisector between M and B that passes through a point c such that $c[i] = (R1_L[i] + R2_L[i])/2$ for all dimensions i for which $B[i] > M[i]$ and $c[j] = (R1_H[i] + R2_H[j])/2$ for all dimensions j for which $B[j] \leq M[j]$. Fig. 5.6 shows two normalized (antipodal) half spaces $H'_{M:B}$ and $H'_{P:A}$. The point c for each half space is also shown. The inequalities (5.3) and (5.4) define the half space $H_{M:B}$ and its normalized half space $H'_{M:B}$, respectively.

$$\sum_{i=1}^{d}(B[i] - M[i]) \cdot x[i] < \sum_{i=1}^{d} \frac{(B[i] - M[i])(B[i] + M[i])}{2} \tag{5.3}$$

$$\begin{aligned}&\sum_{i=1}^{d}(B[i] - M[i]) \cdot x[i] < \\ &\sum_{i=1}^{d}(B[i] - M[i]) \times \begin{cases} \dfrac{(R1_L[i] + R2_L[i])}{2}, & \text{if } B[i] > M[i] \\ \dfrac{(R1_H[i] + R2_H[i])}{2}, & \text{otherwise} \end{cases}\end{aligned} \tag{5.4}$$

Note that the right hand side of the Equation (5.3) cannot be smaller than the right hand side of Equation (5.4). For this reason $H'_{MB} \subseteq H_{MB}$.

Now, we present our pruning rule.

Pruning Rule 5.3.3 *Let R_Q and R_{fil} be two hyper-rectangles. For any point p that lies in $\bigcap_{i=1}^{2^d} H'_{C_i:C'_i}$, $mindist(p, R_Q) > maxdist(p, R_{fil})$ where $H'_{C_i:C'_i}$ is normalized half space between C_i (the i^{th} corner of the rectangle R_{fil}) and its antipodal corner C'_i in R_Q.*

The proof of correctness is non-trivial and is given in Appendix A (see Lemma A.2.5).

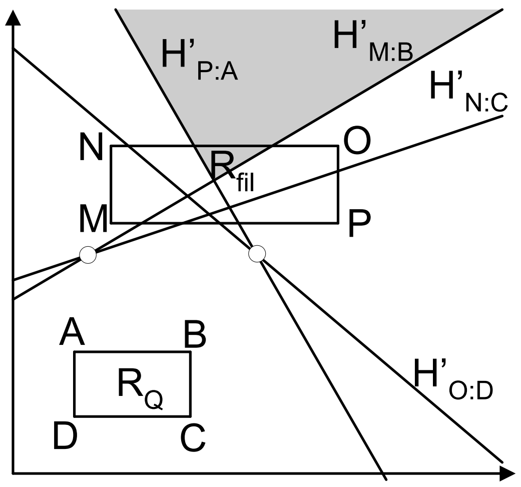


Figure 5.7: Any point in shaded area can never be RNN of any $q \in Q$

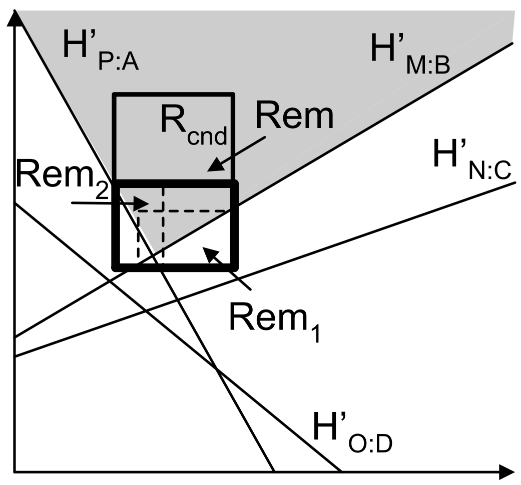


Figure 5.8: Clipping part of the candidate object R_{cnd} that can not be pruned

Consider the example of Fig. 5.7 where the normalized antipodal half spaces are drawn and their intersection is shown shaded. Any point that lies in the shaded area is closer to every point in rectangle R_{fil} than every point in rectangle R_Q.

Note that if R_{fil} and R_Q are the MBRs enclosing all instances of an uncertain object U_i and query object Q, respectively, any instance $u \in U_{j,j\neq i}$ that lies in the pruned region, $\bigcap_{i=1}^{2^d} H'_{C_i:C'_i}$, cannot be RNN of any instance of $q \in Q$ in any possible world. Even if the pruning region partially overlaps with R_{fil}, we can still trim the part of any other hyper-rectangle $R_{U_{j,j\neq i}}$ that falls in the pruned region. It is known that exact trimming becomes inefficient in high dimensional space, therefore, we adopt the loose trimming of R_{cnd} proposed in [TPL04].

The overall half space pruning algorithm that integrates pruning rules 5.3.2 and 5.3.3 is illustrated in Algorithm 7. For each half space, we use the clipping algorithm in [GRSY97] to find a *remnant* rectangle $Rem_i \subseteq R_{cnd}$ that cannot be pruned (lines 4

Algorithm 7 : hspace_pruning (Q, R_{fil}, R_{cnd})

Input: Q: an MBR containing instances of Q ; R_{fil}: the MBR to be used for trimming R_{cnd}: the candidate MBR to be trimmed

Description:

```
1:  $Rem = \varnothing$ /* Remnant rectangle */
2:  for each corner $C_i$ of $R_{fil}$ do
3:    if $Q$ is a point then
4:      $Rem_i = \text{clip}(R_{cnd}, H_{C_i:Q})$/* clipping algorithm [GRSY97] */
5:    else if $Q$ is a hyper-rectangle then
6:      $C'_i$ = antipodal corner of $C_i$ in $Q$
7:      $Rem_i = \text{clip}(R_{cnd}, H'_{C_i:C'_i})$/* clipping algorithm [GRSY97] */
8:    enlarge $Rem$ to enclose $Rem_i$
9:    if $Rem = R_{cnd}$ then
10:     return $R_{cnd}$
11: return $Rem$
```

and 7). After all the half spaces have been used for pruning, we calculate the MBR $Rem \subseteq R_{cnd}$ as the minimum bounding hyper rectangle covering every Rem_i. As such, we trim the original R_{cnd} to Rem.

For better illustration we zoom Fig. 5.7 and show the clipping of a hyper-rectangle R_{cnd} in Fig. 5.8. The algorithm returns Rem_1, Rem_2 (rectangles shown with broken lines) when $H'_{M:B}$ and $H'_{P:A}$ are parameters to the clipping algorithm, respectively. For the half spaces $H'_{N:C}$ and $H'_{O:D}$ the whole hyper-rectangle R_{cnd} can be pruned so the algorithm returns ϕ. The remnant hyper-rectangle Rem is an MBR that encloses Rem_1 and Rem_2. Note that at any stage if the remnant rectangle Rem becomes equal to R_{cnd}, the clipping by other bisectors is not needed so R_{cnd} is returned without further clipping (line 10).

5.3.2 Dominance Pruning

We first give the intuition behind this pruning rule. Fig. 5.9 shows another example of pruning by using pruning rule 5.3.3 in two dimensional space. The normalized half spaces are defined such that if R_{fil} is fully dominated[3] by R_Q in all dimensions then all the normalized antipodal half spaces meet at point F_p as shown in Fig. 5.9. We also observe that for the case when R_{fil} is fully dominated by R_Q, the angle between the half spaces that define the pruned area (shown in grey) is always greater than 90°. Based on these observations, it can be verified that the space dominated by F_p (the dotted-shaded area) can be pruned.

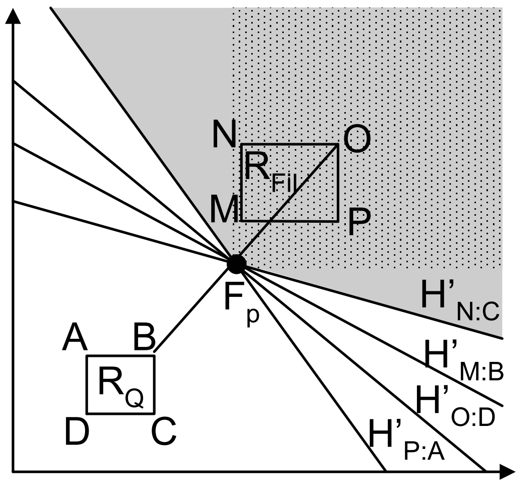


Figure 5.9: Pruning area of half space pruning and dominance pruning

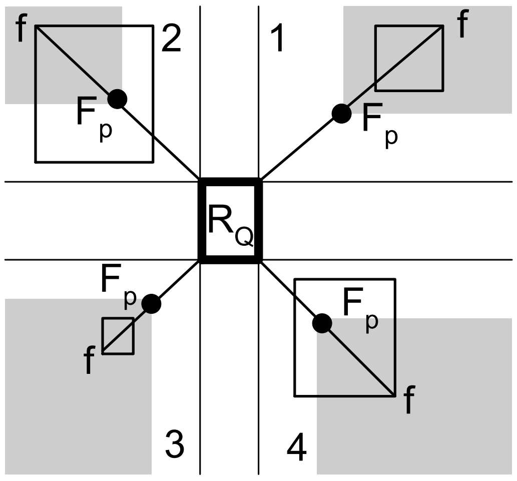


Figure 5.10: Dominance Pruning: Shaded areas can be pruned

Let R_Q be the MBR containing instances of Q. We can obtain the 2^d regions as shown in Fig. 5.10. Let R_{U_i} be an MBR of a filtering object R_{fil} that lies completely in one of the 2^d regions. Let f be the furthest corner of R_{U_i} from R_Q and n be the nearest corner of R_Q from f. The *frontier point* F_p lies at the centre of line joining f and n.

Pruning Rule 5.3.4 *Any instance $u \in U_j$ that is dominated by the frontier point F_p of a filtering object cannot be RNN of any $q \in Q$ in any possible world.*

[3]If every point in R_1 is dominated (dominance relationship as defined in skylines) by every point in R_2 we say that R_1 is fully dominated by R_2.

Formal proof is given in Appendix A (see Lemma A.2.6).

Fig. 5.10 shows four examples of dominance pruning (one in each region). In each partition the shaded area is dominated by F_p and can be pruned. Note that if R_{fil} is not fully dominated by R_Q, we cannot use this pruning rule because the normalized antipodal half spaces in this case do not meet at the same point. For example, the four normalized antipodal half spaces intersect at two points in Fig. 5.7. In general, the pruning power of this rule is less than that of the half space pruning. Fig. 5.9 shows the area pruned by the half space pruning (shaded area) and dominance pruning (dotted area).

The main advantage of this pruning rule is that the pruning procedure is computationally more efficient than the half space pruning, as checking the dominance relationship and trimming the hyper-rectangles is easier.

5.3.3 Metric Based Pruning

Pruning Rule 5.3.5 *An uncertain object R_{cnd} can be pruned if $maxdist(R_{cnd}, R_{fil}) < mindist(R_{cnd}, R_Q)$.*

This pruning approach is the least expensive. Note that it cannot prune part of R_{cnd}, i.e., it either invalidates all the instances of R_{cnd} or does nothing.

5.3.4 Probabilistic Pruning

Note that we did not discuss probability threshold while presenting previous pruning rules. In this section, we present a pruning rule that exploits the probability threshold and embeds it in all previous pruning rules to increase their pruning powers.

A simple exploitation of the probability threshold is to trim the candidate object using previous pruning rules and then prune the object if the accumulative appearance probability of instances within its remnant rectangle is less than the threshold. Next, we present a more powerful pruning rule that is based on estimating an upper bound of the RNN probability of candidate objects.

In previous pruning rules, we prune some area using MBR of a query object R_Q and a filtering object R_{fil}. We observe that the area pruned by using R'_Q and R'_{fil} always contains the area pruned by R_Q and R_{fil} where $R'_Q \subseteq R_Q$ and $R'_{fil} \subseteq R_{fil}$. Fig. 5.11 shows an example. The shaded area is pruned when R'_Q and R_{fil} are used for pruning and the dotted shaded area is pruned when R_Q and R_{fil} are used. Note that this observation also holds for the dominance pruning.

We can use the observation presented above to prune the objects that cannot have RNN probability greater than the threshold. First, we give a formal description of this pruning rule and then we give an example.

Pruning Rule 5.3.6 *Let the instances of Q be divided into n disjoint*[4] *sets $\{Q_1, Q_2, ..., Q_n\}$ and R_{Q_i} be the minimum bounding rectangle enclosing* all *instances in Q_i. Let $\{R_{cnd_1}, R_{cnd_2}$ $, ..., R_{cnd_n}\}$ be the set of bounding rectangles such that each R_{cnd_i} contains the instances of the candidate object that cannot be pruned for Q_i using any of the pruning rules. Let $P^{R_{Q_i}}$ and $P^{R_{cnd_i}}$ be the total appearance probabilities of instances in Q_i and R_{cnd_i}, respectively. If $\sum_{i=1}^{n}(P^{R_{cnd_i}} \cdot P^{R_{Q_i}}) < \rho$, the candidate object can be pruned.*

Pruning rule 5.3.6 computes an upper bound of the RNN probability of the candidate object by assuming that all instances in R_{cnd_i} are RNNs of all instances in Q_i. The candidate object can be safely pruned if this upper bound is still less than the threshold.

Example 5.3.7 *Fig. 5.12 shows MBRs of the query object R_Q and a candidate object R_{cnd} along with their instances (q_1 to q_5 and u_1 to u_4). Assume that all instances within an object have equal appearance probabilities (e.g; $p_{q_i} = 0.2$ for every q_i and $p_{u_i} = 0.25$ for every u_i). Suppose that no part of R_{cnd} can be pruned using R_Q and any filtering object R_{fil} (for better illustration, filtering object is not shown). We prune R_{cnd} using the rectangle R_{Q_1} that is contained by R_Q. This trims R_{cnd} and the remnant rectangle R_1 is obtained. Similarly, R_2 is the remnant rectangle when pruning rules are applied for*

[4]We only require instances of Q to be disjoint. The pruning rule can be applied even when the minimum bounding rectangles R_{Q_i} overlap each other as shown in Fig. 5.12.

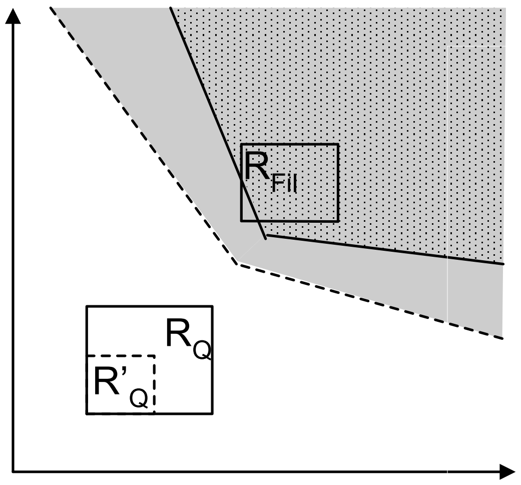


Figure 5.11: Regions pruned by R_Q and its subset R'_Q

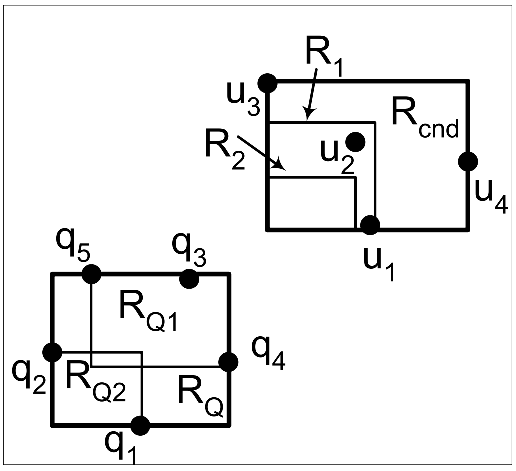


Figure 5.12: Probabilistic pruning

R_{Q_2}. *Note that only the instances in* R_1 *(*u_1 *and* u_2*) can be the RNN of instances in* R_{Q_1} *(*q_3*,* q_4 *and* q_5*). Similarly, no instance can be the RNNs of any instance in* R_{Q_2} *because* R_2 *is empty. So the maximum RNN probability of* R_{cnd} *is* $(0.6 \times 0.5) + (0.4 \times 0) = 0.3$*. If the probability threshold* ρ *is greater than* 0.3*, we can prune* R_{cnd}*. Otherwise, we can continue to trim* R_{cnd} *by using the smaller rectangles contained in* R_{Q_1}*.*

In our implementation, we build an R-tree on query object and the pruning rule is applied iteratively using MBRs of children. For more details, please see Algorithm 11.

Although the smaller rectangles R'_{fil} contained in R_{fil} can also be used, we do not use them because unlike query object there may be many filtering objects. Hence, using the smaller rectangles for each of the filtering objects would make this pruning rule very expensive in practice (more expensive than the efficient verification presented in Section 5.4.3).

5.3.5 Integrating the pruning rules

Algorithm 8 is the implementation of pruning rules 5.3.2 to 5.3.5. Specifically, we apply pruning rules in increasing order of their computational costs (i.e., from pruning rule 5.3.5 to 5.3.2). While simple pruning rules are not as restricting as more expensive ones, they can quickly discard many non-promising candidate objects and save the overall computational time.

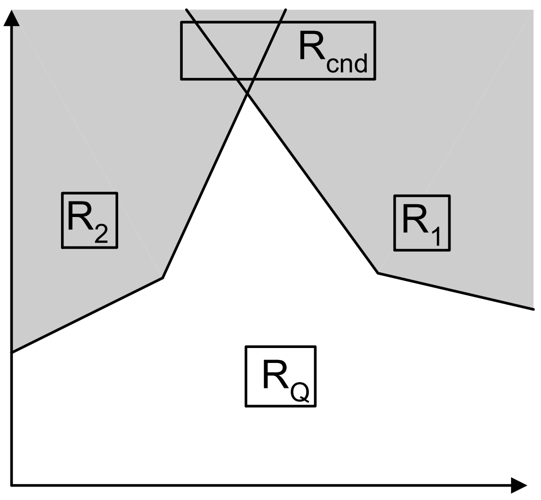


Figure 5.13: R_{cnd} can be pruned by R_1 and R_2

It is important to use *all* the filtering objects to filter a candidate objects. Consider the example in Fig. 5.13. R_{cnd} cannot be pruned by either R_1 or R_2, but will be pruned by considering both of them.

Two subtle optimizations in the algorithm are:

- If $mindist(R_{cnd}, R_{fil}) > maxdist(R_Q, R_{cnd})$ for a given MBR R_{fil}, then R_{fil} cannot prune any part of R_{cnd}. Hence such R_{fil} is not considered for dominance and half space pruning (lines 4-5). However, R_{fil} may still prune some other candidate objects, so we remove such R_{fil} only from a *local* set of filtering object, S_{fil}. This optimization reduces the cost of dominance and half space pruning.

- If the frontier point F_{p_1} of a filtering object R_{fil_1} is dominated by the frontier point F_{p_2} of another filtering object R_{fil_2}, then F_{p_1} can be removed from S_{fil} because the area pruned by F_{p_1} can also be pruned by F_{p_2}. However, note that a frontier point cannot be used to prune its own rectangle. Therefore, before deleting F_{p_1}, we use it to prune rectangle belonging to F_{p_2}. This optimization reduces the cost of dominance pruning.

Algorithm 8 : Prune(Q, S_{fil}, R_{cnd})

Input: R_Q: an MBR containing instances of Q ; S_{fil}: a set of MBRs to be used for trimming R_{cnd}: the candidate MBR to be trimmed

Description:

```
1:  for each $R_{fil}$ in $S_{fil}$ do
2:    if $maxdist(R_{cnd}, R_{fil}) < mindist(R_Q, R_{cnd})$ then /* Pruning rule 5.3.5 */
3:      return $\phi$
4:    if $mindist(R_{cnd}, R_{fil}) > maxdist(R_Q, R_{cnd})$ then
5:      $S_{fil} = S_{fil} - R_{fil}$/* $R_{fil}$ cannot prune $R_{cnd}$ */
    $Rem = R_{cnd}$
6:  for each $R_{fil}$ in $S_{fil}$ do
7:    if $R_{fil}$ is fully dominated by $R_Q$ in a partition $p$ then /* Pruning rule 5.3.4 */
8:      if some part of $Rem$ lies in the partition $p$ then
9:        $Rem$ = the part of $Rem$ not dominated by $F_p$
10:       if $(Rem = \phi)$ then return $\phi$
11: for each $R_{fil}$ in $S_{fil}$ do
12:   $Rem$ = hspace_pruning($R_Q, R_{fil}, Rem$) /* Pruning Rules 5.3.2 and 5.3.3 */
13:   if $(Rem = \phi)$ then return $\phi$
14: return $Rem$
```

5.4 Proposed Solution

In this section, we present our algorithm to find the probabilistic RNNs of an uncertain query object Q. The data is stored in system as follows: for each uncertain object, an R-tree is created and stored on disk that contains the instances of the uncertain object. Each node of the R-tree contains the aggregate appearance probability of the instances in its subtree. We refer these R-trees as *local* R-trees of the objects. Another R-tree is created that stores the MBRs of all uncertain objects. This R-tree is called *global R-tree*.

Algorithm 9 : Answering Probabilistic RNN

Input: Q: uncertain query object; ρ: probability threshold;

Output: all objects that have higher than ρ probability to be RNN of Q

Description:

```
1: Shortlisting: Shortlist candidate and filtering objects (Algorithm 10)
2: Refinement: Trim candidate objects using disjoint subsets of $Q$ and apply pruning rule 5.3.6 (Algorithm 11)
3: Verification: Compute the exact probabilities of each candidate and report results
```

Algorithm 9 outlines our approach. Our algorithm consists of three phases namely Shortlisting, Refinement and Verification. In the following sub-sections, we present the details of each of these three phases.

5.4.1 Shortlisting

In this phase (Algorithm 10), the global R-tree is traversed to shortlist the objects that may possibly be the RNN of Q. The MBR R_{cnd} of each shortlisted candidate object is stored in a set of candidate objects called S_{cnd}. Initially, root entry of the R-tree is inserted in a min-heap H. Each entry e is inserted in the heap with key $maxdist(e, R_Q)$ because a hyper-rectangle that has smaller maximum distance to R_Q is likely to prune a larger area and has higher chances to become the result.

We try to prune every de-heaped entry e (line 5) by using the pruning rules presented in the previous section. If e is a data object and cannot be pruned, we insert it into S_{cnd}. Otherwise, if e is an intermediate or leaf node, we insert its children c into heap H with key $maxdist(c, R_Q)$. Note that an entry e can be removed from S_{fil} (line 9) if at least one of its children is inserted in S_{fil} because the area pruned by an entry e is always contained by the area pruned by its child (a direct implication of Lemma A.2.5 in Appendix A).

Algorithm 10 : Shortlisting

```
1: $S_{fil} = \emptyset$, $S_{cnd} = \emptyset$
2: Initialize a min-heap $H$ with root entry of Global R-Tree
3: while H is not empty do
4:     de-heap an entry $e$
5:     if $(Rem = \text{prune}(R_Q, S_{fil}, e)) \neq \phi$ then
6:         if $e$ is a data object then
7:             $S_{cnd} = S_{cnd} \cup \{e\}$
8:         else if $e$ is a leaf or intermediate node then
9:             $S_{fil} = S_{fil} - \{e\}$
10:            for each data entry or child $c$ in $e$ do
11:                insert $c$ into $H$ with key $maxdist(c, R_Q)$
12:                $S_{fil} = S_{fil} \cup \{c\}$
```

5.4.2 Refinement

In this phase (Algorithm 11), we refine the set of candidate objects by using pruning rule 5.3.6. More specifically, we descend into the R-tree of Q and trim each candidate object R_{cnd} against the children of Q and apply pruning rule 5.3.6.

Let P^R be the aggregate probability of instances in any hyper-rectangle R. At this stage $P^{R_{cnd}}$ of a candidate object may be less than one because R_{cnd} might have been trimmed during shortlisting phase. We can prune R_{cnd} if upper bound RNN probability of a candidate object $MaxProb = P^{R_{cnd}}$ is less than ρ (line 3).

We use a max-heap that stores entries in form (e, R, key) where e and R are hyper-rectangles containing instances of Q and R_{cnd}, respectively. key is the maximum probability of instances in R to be the RNNs of instances in e (i.e; $key = P^e \cdot P^{R_{cnd}}$). We initialize the heap by inserting (Q, R_{cnd}, $MaxProb$) (line 5). For each de-heaped entry (e, R, p), we trim the hyper-rectangle R against e by S_{fil} and store the trimmed rectangle in Rem (line 8). The upper bound RNN probability $MaxProb$ is updated to

Algorithm 11 : Refinement

Description:

```
1: for each $R_{cnd}$ in $S_{cnd}$ do
2:   if $(MaxProb = P^{R_{cnd}}) < \rho$ then
3:     $S_{cnd} = S_{cnd} - R_{cnd}$; continue;
4:   Initialize a max-heap H containing entries in form $(e, R, key)$
5:   insert $(Q, R_{cnd}, MaxProb)$ into $H$
6:   while H is not empty do
7:     de-heap an entry $(e, R, p)$
8:     $Rem$ = Prune $(e, S_{fil}, R)$
9:     $MaxProb = MaxProb - p + (P^e \cdot P^{Rem})$
10:    if $MaxProb < \rho$ then
11:      $S_{cnd} = S_{cnd} - R_{cnd}$; break;
12:    if $(P^{Rem} > 0)$ AND ($e$ is an intermediate node or leaf) then
13:      for each child $c$ of $e$ do
14:        insert $(c, Rem, (P^c \cdot P^{Rem}))$ into H
```

$MaxProb - p + (P^e \cdot P^{Rem})$. Recall that $p = P^e \cdot P^R$ was inserted with this entry assuming that all instances in R are RNNs of all instances in e. After we trim R using e (line 8), we know that only the instances in Rem can be RNNs of e. That is the reason we subtract p from $MaxProb$ and add $(P^e \cdot P^{Rem})$.

At any stage, if the $MaxProb < \rho$ the candidate object can be pruned. Otherwise, an entry $(c, Rem, (P^c.P^{Rem}))$ is inserted into the heap, for each child c of e. Note that if the trimmed hyper-rectangle does not contain any instance then P^{Rem} is zero and we do not need to insert children of e in the heap for such Rem.

Recall that every node in local R-tree stores the aggregate appearance probability of all instances in its sub-tree which makes computation of aggregate probability cheaper.

5.4.3 Verification

The actual probability of a candidate object R_{cnd} to be the RNN of Q is the sum of probabilities of every instance $u_i \in R_{cnd}$ to be the RNN of every instance q of Q . To compute the probability of an instance u_i to be RNN of q, we have to find, for each uncertain object U, the accumulative appearance probability of its instances that have smaller distance to u_i than $dist(q, u_i)$ (Equation (5.2)). A straight forward approach is to issue a range query for every $u_i \in R_{cnd}$ centred at u_i with range set as $dist(q, u_i)$ and then compute the accumulative appearance probability of instances of each object that are returned. However, this approach requires $\mid Q \mid \times \mid R_{cnd} \mid$ number of range queries where $\mid Q \mid$ and $\mid R_{cnd} \mid$ are number of instances in Q and R_{cnd}, respectively. Below, we present an efficient approach that issues only one global range query to compute the exact RNN probability of a candidate object.

Finding range of the global range query

Let R_{fil} be an MBR containing instances of a filtering object. An instance u_i has zero probability to be RNN of an instance q if $dist(u_i, q) > maxdist(u_i, R_{fil})$. So the range of a range query for u_i centred at u_i is minimum of $maxdist(u_i, R_Q)$ and $maxdist(u_i, R_{fil})$ for every R_{fil} in S_{fil}.

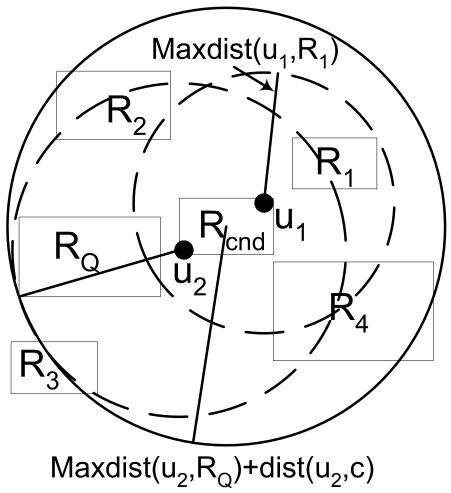


Figure 5.14: Finding the range of the global query

Consider the example of Fig. 5.14 where the range of queries centred at u_1 and u_2 are $maxdist(u_1, R_1)$ and $maxdist(u_2, R_Q)$, respectively (circles with broken lines).

We want to reduce multiple range queries to a single range query centred at the centre of R_{cnd} with a *global range* r such that all instances required to compute RNN probability of every candidate instance $u_i \in R_{cnd}$ are returned. Let r_i be the range of the range query of u_i computed as described above. The global range r is $\max(r_i + dist(u_i, c))$ for every $u_i \in R_{cnd}$ where c is the centre of R_{cnd}. In the example of Fig. 5.14, the global range is $r = maxdist(u_2, R_Q) + dist(u_2, c)$ as shown in the figure (solid circle). Note that this range ensures that all the instances required to compute RNN probability of both u_1 and u_2 lie within this range.

Computing the exact RNN probability of R_{cnd}

We issue a range query on global R-tree with range r as computed above. For each returned object U_i, we issue a range query on the local R-tree of U_i to get the instances that lie within the range and then create a list L_i containing all these instances. We sort the entries in each list L_i in ascending order of their distances from u_{cnd}.

The list L_Q for the instances of query object Q is shown in Fig. 5.15. Each entry e contains two values (d, p) such that d is distance of e from u_{cnd} and p is the appearance probability of the instance e. The lists for other objects are slightly different in that each entry e contains two values (d, P) where P is the *accumulative* appearance probability of all the instances that appear in the list before e. In other words, given an entry (d, P), the total appearance probability of all instances (in this list) that have smaller distance than d is P.

Given these lists, we can quickly find the accumulative appearance probability of all instances of any uncertain object that lie closer to u_{cnd} than a query instance q_i. The example below illustrates the computation of exact probability of a candidate instance u_{cnd}.

Example 5.4.1 *Fig. 5.15 shows the lists of query object Q and three uncertain objects*

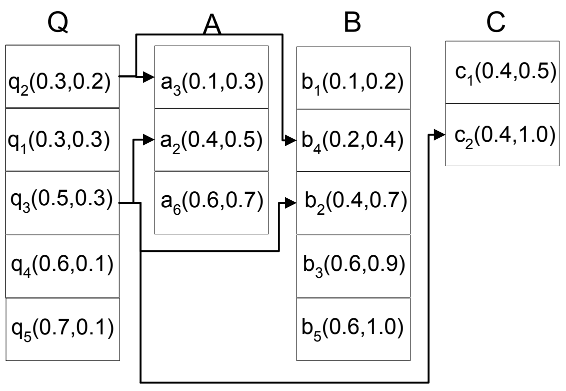


Figure 5.15: lists sorted on distance from a candidate instance u_{cnd}

A, B and C. The lists are sorted on their distances from the candidate instance u_{cnd}. We start the computation from the first entry q_2 in Q and compute $RNN_{q2}(u_{cnd})$. The distance d_{q_2} is 0.3. We do a binary search on A, B and C to find an entry in each list with largest d smaller than d_{q_2}. Such entries are $a_3(0.1, 0.3)$ and $b_4(0.2, 0.4)$ in lists A and B, respectively. No instance is found in C. Hence, the sum of appearance probabilities of instances of B that have distance from u_{cnd} smaller than d_{q_2} is 0.4, similarly for A it is 0.3. Given both q_2 and u_{cnd} appear in a world, the probability of u_{cnd} to be RNN of q_2 is obtained from Equation (5.2) as $(1 - 0.4)(1 - 0.3) = 0.42$. The probability of u_{cnd} to be RNN of q_2 in any possible world is $0.42(p_{q_2} \times p_{u_{cnd}})$.

Similarly the next entry in Q is processed and $RNN_{q_1}(u_{cnd})$ is computed which is again 0.42 because its distance from u_{cnd} is the same. $RNN_{q_3}(u_{cnd})$ is zero because the binary search on C gives an entry (d, P) where $P = 1$ (all instances of C have smaller distance to u_{cnd} then d_{q_3}). Note that, we do not need to compute the RNN probabilities of u_{cnd} against remaining instances q_4 and q_5 because their distances from u_{cnd} are larger than d_{q_3} and $RNN_{q_3}(u_{cnd}) = 0$. Also note that the area to be searched in any list L_i by binary search becomes smaller for the processing of next query instance.

The above example illustrates the probability computation of an instance u_{cnd} to be the RNN of all instances in Q. We repeat this for every instance $u_{cnd} \in R_{cnd}$ to compute the RNN probability of the candidate object. Next, we present some optimizations that improve the efficiency of verification phase.

Optimizations

Our proposed optimizations bound the minimum and maximum RNN probabilities and verify the objects that have the minimum probability greater than or equal to the threshold. Similarly, the objects that have the maximum probability less than the threshold are deleted. Below, we present the details of the proposed optimizations.

a) Bounding RNN probabilities using R_Q:

Recall that, for each candidate object R_{cnd}, a global range query is issued and for each object U_i within the range a list L_i is created containing the instances of U_i lying within the range. Just before we sort these lists, we can approximate the maximum and minimum RNN probability of the candidate object based on the following observations.

Let c be the centre and d be the diagonal length of R_{cnd} and a_i be some instance in list A. Every $u_{cnd} \in R_{cnd}$ is always closer to a_i than every $q_i \in Q$ if $mindist(R_{cnd}, R_Q) > dist(a_i, c) + d/2$. Similarly, every u_{cnd} would always be further from a_i than every $q_i \in Q$ if $maxdist(R_{cnd}, R_Q) < dist(a_i, c) - d/2$. Consider the example of Fig. 5.16, every point in R_{cnd} is always closer to a_1 than any point in R_Q. Similarly, every point in R_{cnd} is always further from a_2 than it is from any point in R_Q.

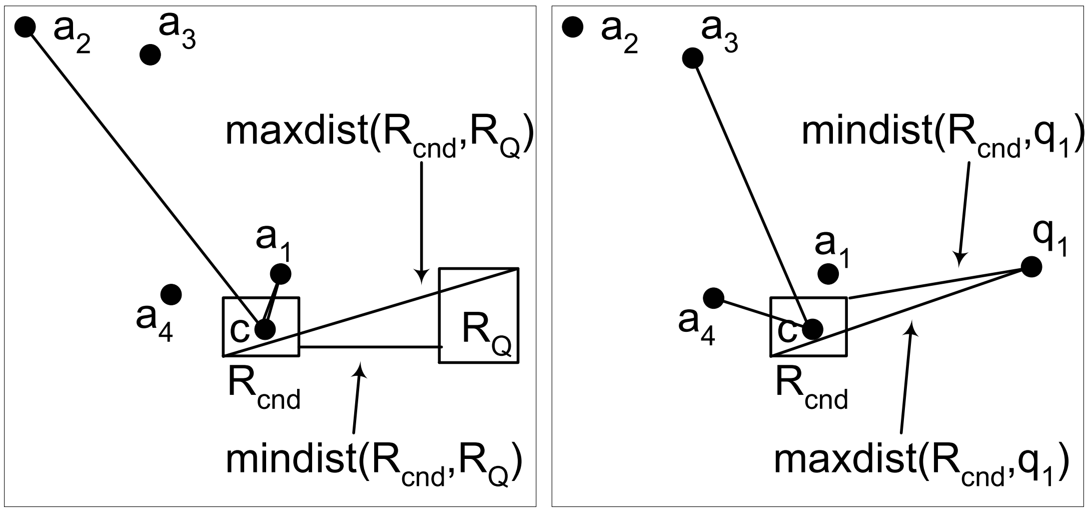


Figure 5.16: Bounding lower and upper bound RNN probabilities

Based on the above observations, for every object, we can accumulate the appearance probabilities of all the instances u such that every u_{cnd} is always closer to (or further from) u than every q_i. More specifically, we traverse each list L_i and accumulate the appearance probabilities of every instance u_i for which $mindist(R_{cnd}, R_Q) > dist(u_i, c) + d/2$ and store the accumulated probabilities in P_i^{near}. Similarly, the accumulated appearance probabilities of every instance u_j for which $maxdist(R_{cnd}, R_Q) < dist(u_j, c) - d/2$ is stored in P_i^{far}. Then the maximum RNN probability of any instance u_{cnd} is $p_{cnd}^{max} = \prod_{\forall L_i}(1 - P_i^{near})$. The minimum probability of any instance u_{cnd} to be RNN of Q is $p_{cnd}^{min} = \prod_{\forall L_i}(P_i^{far})$ because P_i^{far} is the total probability of instances that are definitely farther. So we assume that all other instances are closer to u_{cnd} than q_i and this gives us the minimum RNN probability.

Let $P^{R_{cnd}}$ be the aggregate appearance probability of all the instances in R_{cnd} then R_{cnd} can be pruned if $P^{R_{cnd}} \cdot p_{cnd}^{max} < \rho$. Similarly, the object can be reported as answer if $P^{R_{cnd}} \cdot p_{cnd}^{min} \geq \rho$.

b) Bounding RNN probabilities using instances of Q:

If an object R_{cnd} cannot be pruned or verified as result at this stage, we try to make a better estimate of p_{cnd}^{min} and p_{cnd}^{max} by using instances within Q. Note that every $u_{cnd} \in R_{cnd}$ is always closer to a_i than a query instance q_i if $mindist(R_{cnd}, q_i) > dist(a_i, c) + d/2$. Similarly, every u_{cnd} would always be further from a_i than q_i if $maxdist(R_{cnd}, q_i) < dist(a_i, c) - d/2$. Consider the example of Fig. 5.16 where every point in R_{cnd} is closer to both a_1 and a_4 than q_1. Similarly, every point in R_{cnd} is further from both a_2 and a_3 than it is from q_1.

To update p_{cnd}^{max}, we first sort every list in ascending order of $dist(c, u)$ where $dist(c, u)$ is already known (returned by global range query). Then, the list L_Q is sorted in ascending order of the $mindist(R_{cnd}, q_i)$. Then for each q_i in ascending order, we conduct a binary search on every list L_i and find the entry $e(d, P)$ with greatest d in the list that is less than $mindist(R_{cnd}, q_i) - d/2$. The probability P of this entry is accumulated appearance probability P_i^{near} of all the instances a_i such that every u_{cnd} is always closer to a_i than q_i. Then the maximum probability of any instance $u_{cnd} \in R_{cnd}$ to be the RNN of q_i is $p^{max}_{i_{cnd}} = \prod_{\forall L_i}(1 - P^{near}_i)$. We do such binary searches for every q_i in the list and $p^{max}_{cnd} = \sum_{\forall q_i \in Q} p^{max}_{i_{cnd}}$.

The update of p^{min}_{cnd} is similar except that the list L_Q is sorted in ascending order of $maxdist(R_{cnd}, q_i)$ and the binary search is conducted to find the entry $e(d, P)$ with the greatest d that is smaller than $maxdist(R_{cnd}, q_i)+d/2$. The total appearance probabilities of all instances in L_i that are always farther from every u_{cnd} than q_i is $P^{far}_i = (1 - P)$. Finally, $p^{min}_{i_{cnd}} = \prod_{\forall L_i}(P^{far}_i)$ and $p^{min}_{cnd} = \sum_{\forall q_i \in Q} p^{min}_{i_{cnd}}$.

After updating p^{max}_{cnd} and p^{min}_{cnd}, we delete the candidate objects for which $P^{R_{cnd}} . p^{max}_{cnd} < \rho$. Similarly, a candidate object is reported as answer if $P^{R_{cnd}} \cdot p^{min}_{cnd} \geq \rho$.

c) Early stopping:

If an object R_{cnd} is not pruned by the above mentioned estimation of maximum and minimum RNN probabilities then we have to compute exact RNN probabilities (as described in Section 5.4.3) of the instances in it. By using the maximum and minimum RNN probabilities, it is possible to verify or invalidate an object without computing the exact RNN probabilities of all the instances. We achieve this as follows; We sort all the instances in R_{cnd} in descending order of their appearance probabilities. Assume that we have computed the exact RNN probability $RNN_Q(u)$ of first i instances. Let P be the aggregate appearance probabilities of these first i instances and P_{RNN} be the sum of their $RNN_Q(u)$. At any stage, an object can be verified as answer if $P_{RNN} + (1 - P).p^{min}_{cnd} \geq \rho$. Similarly, an object can be pruned if $P_{RNN} + (1 - P).p^{max}_{cnd} < \rho$.

Note that $(1 - P).p^{min}_{cnd}$ is the minimum probability for the rest of the instances to be the RNN of Q. Similarly, $(1 - P).p^{max}_{cnd}$ is the maximum probability for the remaining instances to be the RNN.

5.5 Experiment Results

In this section we evaluate the performance of our proposed approach. All the experiments were conducted on Intel Xeon 2.4 GHz dual CPU with 4 GBytes memory. The node size of each local R-tree is $1K$ and that of global R-tree is $2K$. *We measured both the I/O and CPU time and I/O cost is around 1-5% of the total cost for all experiments.* Hence, for clarity of experiment figures, we display the average total cost per query. We used both synthetic and real data sets.

Table 5.2: System parameters

Parameter	**Range**
Probability threshold (ρ)	0.1, 0.3, **0.5**, 0.7, 0.9
Number of objects ($\times$1000)	2, 4, **6**, 8, 10
Maximum number of instances in an object	200, 400, **600**, 800, 1000
Maximum width of hyper-rectangle	1%, **2%**, 3%, 4%
Distribution of object centres	**Uniform**, Normal
Distribution of instances	**Uniform**, Normal
Appearance probability of instances	**Uniform**, Normal

Table 5.2 shows the specifications of the synthetic data sets we used in our experiments and the defaults values are shown in bold. First the centres of the uncertain objects were created (uniform or normal distribution) and then the instances for each object (uniform or normal distribution) were created within their respective hyper-rectangles. The width of the hyper-rectangle in each dimension was set from 0 to w% (following uniform distribution) of the whole space and we conducted experiments for w changed from 1 to 4. The appearance probabilities of instances were generated following either uniform or normal distribution. Our default synthetic data set contains approximately 1.8 Million instances ($\frac{6000\times600}{2}$). Similar to [TCX+05], the query object follows same distribution as the underlying data set.

The real data set[5] consists of 28483 zip codes obtained from 40 states of United States. Each zip code represents an object and the *address blocks* within each zip code are the instances. The data source provides address ranges instead of individual addresses and

[5]http://www.census.gov/geo/www/tiger/

we use the term *address block* for a range of addresses along a road segment. The address block is an instance in our data set that lies at the middle of the road segment with the appearance probability calculated as follows; Let n be the number of total addresses in a zip code and m be the number of addresses in the current address block then the appearance probability of the current address block is m/n. The real data set consists of 11.24 Million instances and the maximum number of instances (address blocks) in an object (Sanford, North Carolina) were 5918.

5.5.1 Comparison with other possible solutions

We devise a naïve algorithm and a sampling based approximate algorithm to better understand the performance of our algorithm. More specifically, in the naïve algorithm, we first shortlist the objects using our pruning rule 5.3.5 (e.g; any object R_{cnd} can be pruned if $mindist(R_{cnd}, R_Q) > maxdist(R_{cnd}, R_{fil})$). Then, we verify the remaining objects as follows. For each pair (u_i, q_i), we issue a range query centred at u_i with range $dist(u_i, q_i)$ and compute the RNN probability of the instance u_i against the query instance q_i using the Equation (5.2). Finally, the Equation (5.1) is used to compute the RNN probability of the object.

In sampling based approach, we create a few sample possible worlds before starting the computation. More specifically, a possible world is created by randomly selecting one instance from each uncertain object. For each possible world, we create an R-tree (node size $2K$) that stores the instances of the possible worlds. This reduces the problem of finding probabilistic RNNs to conventional RNNs. For each possible world, we compute the RNNs using TPL [TPL04] that is the best-known RNN algorithm for multidimensional data. Let n be the number of possible worlds evaluated and m be the number of possible worlds in which an object R_{cnd} is returned as RNN, then R_{cnd} is reported as answer if $m/n \geq \rho$. The costs shown do not consider the time taken in creating the possible worlds. Note that this algorithm provides only approximate results. For real data set, the accuracy varies from 60% to 75%.

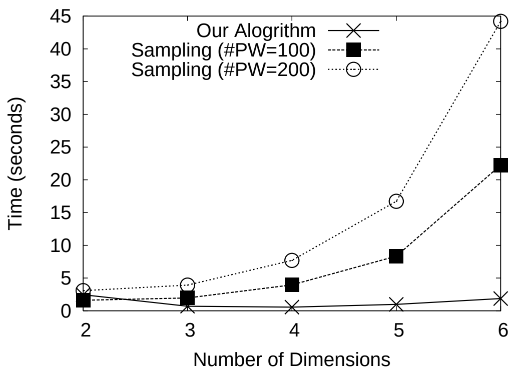


Figure 5.17: Overall cost

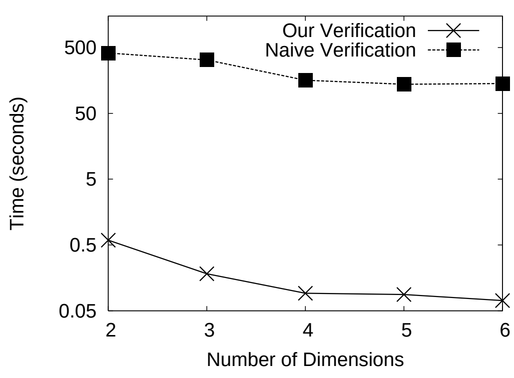


Figure 5.18: Verification cost

Naïve algorithm appeared to be too slow (average query time from 7 minutes to 2 hours) so we show its computation time only when comparing our verification phase in Fig. 5.18.

Fig. 5.17 compares our approach with the sampling based approximate approach (for 100 and 200 possible worlds) on synthetic data set. In two dimensional space, our algorithm is comparable with the sampling algorithm that returns approximate answer. On the other hand, the Fig. 5.17 shows that our algorithm is more efficient for higher dimensions and scales better. The cost for our algorithm first decreases as the number of dimensions increase and then it starts increasing. The reason is that for low dimensional space, the data is more dense and the verification phase cost dominates the pruning phase cost. On the other hand, for high-dimensional space, the data is sparse and while the verification is cheaper the pruning phase is expensive (e.g; greater number of bisectors required to prune the space).

In Fig. 5.18, we compare the verification cost of our algorithm with the verification cost of naïve algorithm. The costs shown are verification costs per candidate object. Our proposed verification is three orders of magnitude faster than the naïve verification.

5.5.2 Performance on real data set and effect of data distribution

Fig. 5.19 compares the performance of our algorithm against the sampling based approximate algorithm on real data set for probability threshold changed from 0.1 to 0.9. For sampling based algorithm, the costs are shown for the evaluation of 100 and 200 possible worlds. Our algorithm performs better than the approximate sampling based algorithm for larger threshold.

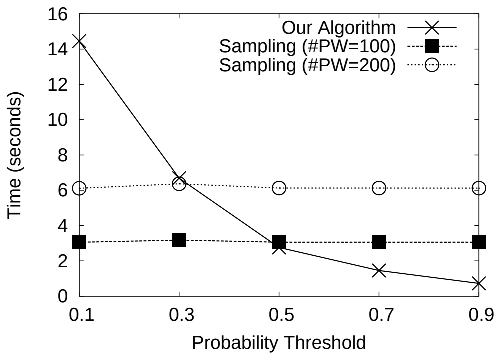


Figure 5.19: Comparison on real data set

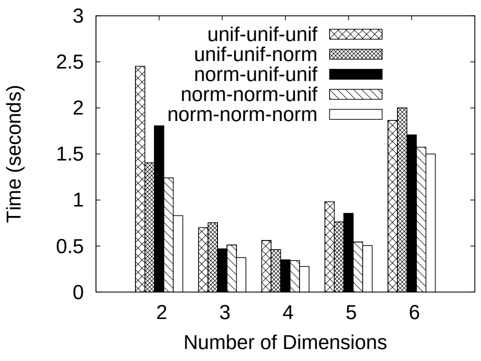


Figure 5.20: Effect of data distribution

Note that although the accuracy may vary, the cost of sampling algorithm does not change with the change in threshold, underlying data distribution (as noted in [TPL04]), width of hyper-rectangle or number of instances in each object. Moreover, the cost of sampling algorithm increases linearly with the number of possible worlds evaluated. For this reason, now we focus on the performance evaluation of only our proposed algorithm.

Fig. 5.20 shows the performance of our algorithm for different data distributions. The legend shows data distributions in form dist1_dist2_dist3 where dist1 is the distribution of the object centres, dist2 is the distribution of instances within the objects and dist3 is the distribution of appearance probability. For example, norm_norm_unif shows the result for the data such that the centres of objects and instances are normally distributed with appearance probability following uniform distribution. The performance of our algorithm on non-uniform data is better than the uniform data as can be observed from Fig. 5.20. This is mainly due to two reasons. Firstly, we observe that the number of candidates in S_{cnd} is smaller after the pruning phase if the data is non-uniform. Secondly, if the probability distribution is not uniform the verification phase is faster because we sort the instances in descending order of their appearance probabilities and this lets us validate or invalidate an object earlier.

5.5.3 Effect of data size

In Fig. 5.21, we increase the maximum number of instances in each object from 200 to 1000. The performance degrades as the number of instances increase. Although the increase in number of instances does not have significant effect on pruning phase, the verification phase becomes more expensive if each object has greater number of instances. Also observe that the cost does not change significantly for higher dimensions because in high dimensional space, the pruning phase cost is dominant which is not affected significantly by the number of instances

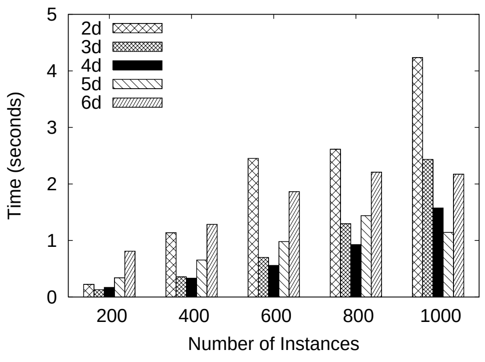


Figure 5.21: Effect of number of instances in each object

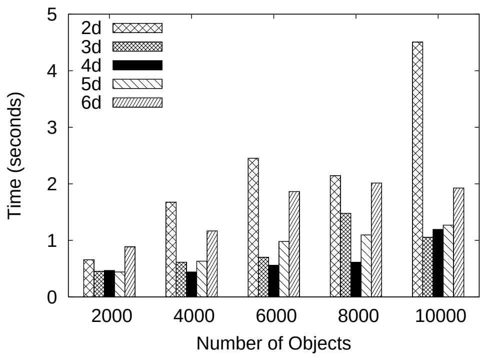


Figure 5.22: Effect of number of objects in the data set

Fig. 5.22 evaluates the performance of our algorithm with increasing number of objects in the data set. The computation cost increases with increase in number of objects mainly due to the increased verification cost because larger number of objects (and in effect instances) are returned by the global range query.

5.5.4 Effect of probability threshold and width of hyper-rectangle

Fig. 5.23 shows the effect of probability threshold. The algorithm performs better as the probability threshold ρ increases because fewer number of candidate objects pass the pruning phase and require the verification. The effect is more significant in lower dimensions because for low dimensions the verification cost dominates the overall cost.

In Fig. 5.24, we change width of each hyper-rectangle and study the performance of

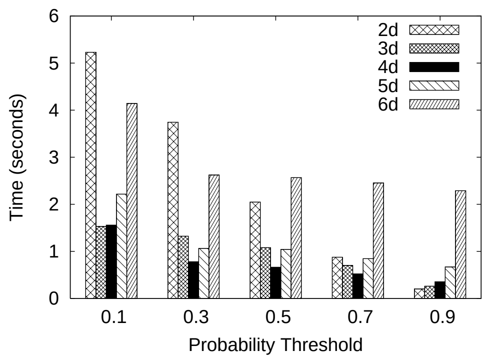


Figure 5.23: Effect of probability threshold

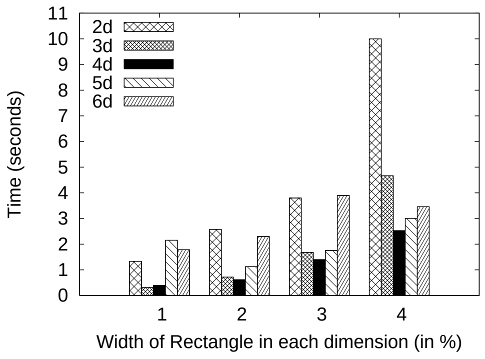


Figure 5.24: Effect of width of hyper-rectangles

our algorithm. The performance degrades in low-dimensional space due to larger overlap of objects with each other and the query object. The effect in higher dimensions is not as significant as in low-dimensional space.

5.5.5 Evaluation of different phases

In this section, we study the effect of our pruning phases. More specifically, we compare the number of candidates after first phase (shortlisting), second phase (refinement), optimization (of the verification phase) and the number of objects in final result. Fig. 5.25 shows the number of candidates after each phase. The number of candidates after *shortlisting* is from 10-20 and the *refinement* phase reduces the number to less than its half. The optimization presented in the verification phase prunes more objects in high-dimensional space because in low-dimensional space due to larger volume of MBRs, most of the MBRs of remaining candidates overlap with the query object. Hence the optimizations are more useful for higher dimensions.

Fig. 5.26 shows the time taken by each of the pruning phase. Our proposed optimization takes very small amount of time and is quite useful especially for high-dimensional data. Verification phase is the dominant cost for low-dimensional queries and the pruning phases (shortlisting and refinement) dominate the overall cost for high-dimensional queries. Note that log scale is used for y-axis.

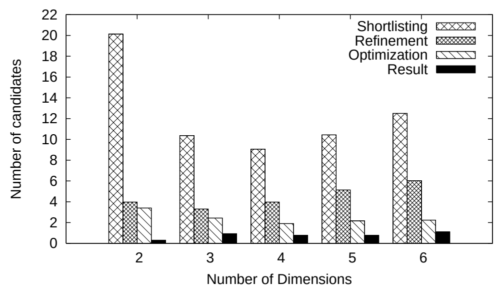


Figure 5.25: Number of objects in S_{cnd} after each phase

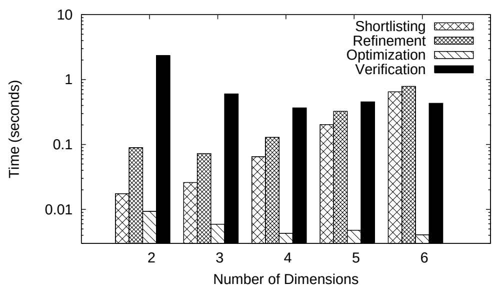


Figure 5.26: Computational time taken by each phase

5.5.6 Effectiveness of pruning rules

Pruning rule 5.3.6 is used in phase 2 (refinement) of our algorithm and uses the other pruning rules to estimate the maximum probability. Its effectiveness can be observed in Fig. 5.25 by comparing the number of objects after *shortlisting* and *refinement* phases.

Fig. 5.27 shows the effectiveness of other pruning rules. We observed that the dominance pruning rule prunes fewer objects than the simple distance based pruning rule 5.3.5. However, the dominance pruning can prune some objects that cannot be pruned by the simple pruning rule because the dominance pruning rule can trim part of the candidate objects.

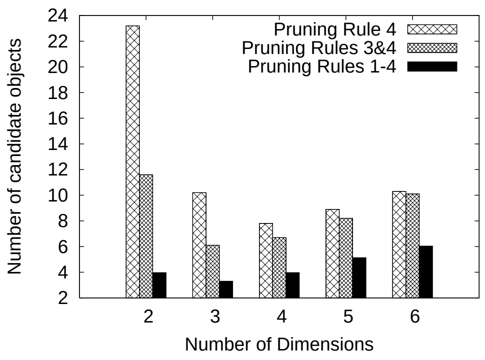


Figure 5.27: Effectiveness of pruning rules

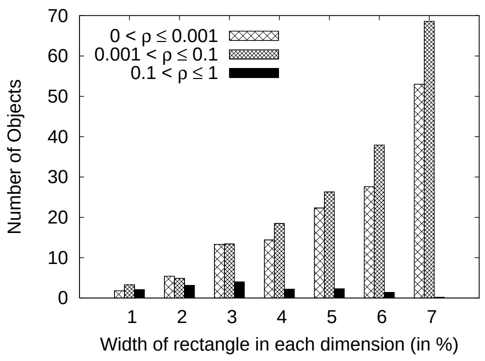


Figure 5.28: Effect of width of hyper-rectangles

Fig. 5.27 shows the number of candidates after *refinement* phase of our algorithm when a combination of pruning rules is used. More specifically, we compare the number of objects in S_{cnd} when only the pruning rule 5.3.5 is used, the dominance pruning is used along with pruning rule 5.3.5, and when all pruning rules from 1 to 4 are used. Since pruning rule 5.3.6 uses the other underlying pruning rules, it is enabled for all above mentioned settings. The half space pruning significantly reduces the number of candidate objects and the effectiveness of dominance pruning is more significant for the low-dimensional data.

5.5.7 Effect of hyper-rectangle width on the size of result

We note that if the hyper-rectangles of objects largely overlap each other, the probabilistic reverse nearest neighbor queries are not very meaningful. In other words, there would be no objects satisfying some reasonable probability threshold (a value that can be considered significant). Fig. 5.28 shows the number of objects that satisfy different probability thresholds. The width of hyper-rectangle in each dimension is changed from 1% to 7% and the results are shown for two dimensional space. It can be observed that with large overlap in rectangles, more and more objects satisfy very small probability threshold constraint. On the other hand, there are very few or no object at all that have greater than 0.1 probability to be the RNN.

5.6 Summary

In this chapter, we formalize probabilistic reverse nearest neighbor query that is to retrieve the objects from the uncertain data that have higher probability than a given threshold to be the RNN of an uncertain query object. We develop an efficient algorithm based on various novel pruning approaches that solves the probabilistic RNN queries on multidimensional uncertain data. The experimental results demonstrate that our algorithm is even more efficient than a sampling-based approximate algorithm for most of the cases and is highly scalable.

Chapter 6

Continuous Monitoring of Moving Range Queries

In this chapter, we present algorithms to continuously monitor moving range queries in Euclidean space as well as in spatial networks. Our research reported in this chapter also appeared in [CBL+10, CBL+11].

6.1 Overview

We consider a set O of objects, a query point q and a positive value r. We use $dist(o, q)$ to denote the distance between an object $o \in O$ and the query q. A distance based range query returns every object $o \in O$ that lies within distance r of the query location q, i.e., every object such that $dist(o, q) \leq r$. Our main focus in this chapter is on Euclidean distance based range queries. Since the search space around the query is a circle in this case, such queries are also called circular range queries. We also consider the case when $dist(o, q)$ is the network distance between o and q (e.g., queries moving in a road network).

Another variation of the range query, which we term "rectangular range query" (also called *window query*), returns the objects that lie within a rectangle around the query location. Distance based range queries and rectangular range queries are inherently different and have different applications. When clear by context, we use the term *range query* to refer to the distance based range queries.

Due to availability of inexpensive position locators, cheap network bandwidth and mobile devices with computation and storage capabilities, location based services are gaining increasing popularity. Consequently, continuous monitoring of spatial queries has received significant research attention in the past few years [CHC04, HXL05, MHP05, BJKS02, CLZ$^+$09, ZZP$^+$03, GL04, TP02].

We study the continuous monitoring of moving range queries over static data objects, i.e., a scenario where the queries are constantly moving whereas the data objects do not change their locations. Such scenario has many interesting applications. Consider the example of a family travelling by car. Suppose they need to reach their final destination by a certain time and only have up to 90min available for lunch. They may want to continuously monitor restaurants within 10km of their current location so that they can choose a restaurant that serves their favorite meals, and will not take more than 15 min to reach. As another example, a bomber plane might want to continuously monitor the enemy targets (e.g., airport, arms depot) that are within its attack range.

We next discuss two models to monitor spatial queries.

Client-server model. In this model, the clients issue queries and the central server is responsible for the computation of these queries. For example, a person walking down the street may issue a query to his mobile service provider to continuously report the coffee shops within 1km of the issuer's location. It may be assumed that the server processes the query in the main-memory, i.e., the data objects are stored in the main-memory along with other relevant information needed to efficiently update the results. However, such systems require that the server continuously maintains this information in the main-memory in order to provide the service.

We neither require that the data objects are stored in the main-memory nor do we maintain any query information in the main-memory. One advantage of this is that the service can be run on-demand. Since the objects are stored in the secondary memory and no main-memory information is maintained, the server can go to sleep mode if there is no query. When a query arrives, the server computes the results and the *safe zone*, which are then sent to the client. The safe zone is an area such that the reported results are valid as long as the client (i.e., query) remains within the safe zone. A query that leaves its safe zone sends an update request. The server updates the safe zone and the results, and sends them back to the client.

Local computation model. In the first application mentioned above, the car may have a GPS navigation system with points of interest (e.g., restaurants) stored in its memory card. Since the navigation systems have limited main-memory and computational capacity, it may be challenging to compute the results of the range query whenever the query changes its location (the car is continuously moving). Our proposed approach returns a safe zone which guarantees that the results of the query do not change as long as the query remains within the safe zone. The safe zone is updated efficiently when the query leaves the safe zone. Our experimental results demonstrate that the overhead to compute the safe zone is small compared to the cost of the range query. This enables our framework to work effectively on the devices with limited main-memory and computation power. We next highlight some advantages of our proposed approach.

a. The computation of the safe zone reduces the overall computation time because the query needs to be re-evaluated only when it leaves the safe zone. Our experiments indicate that the cost of computing the safe zone is small compared to the cost of the range query.

b. Although the shape of the safe zone may be arbitrarily complex, we can still efficiently check whether the query lies within it. If the query is based on network distance, the safe zone itself is a small network that is a subset of the original network and it has to be determined whether the query lies within the safe zone or not. For the circular range queries, we utilize the fact that the safe zone only depends on the so-called *guard* objects. Checking whether the query lies within the safe zone takes k distance computations, where k is the number of guard objects. Our experimental results demonstrate that the average number of guard objects is around 5. This makes our proposed approach applicable for the clients that have limited computational power. We also present a theoretical analysis and give an upper bound on the expected number of guard objects for the queries with the diameter of the safe zone no more than a constant times its expected value.

c. We do not require the data objects to be stored in the main-memory, which allows our approach to work on systems with limited main-memory (e.g., GPS navigation systems).

d. When an update request is received, the server computes the new safe zone and the results for the circular range queries. After updating the results, the server only sends new information to the clients. For example, if the client was informed that an object o_i is within its range, the object o_i is not sent again in updated results if it still lies within the range. If in the future such object o_i ceases to be within the range, the client is informed that o_i is out of the range. Our experimental results demonstrate that this significantly reduces the amount of data transmitted from the server to the clients.

e. In the client-server paradigm, our proposed approach does not require the server to maintain or record any information related to the queries, yet it efficiently updates the safe zones. This enables the server to run this service on-demand.

Note that some computation models require queries to get registered at the server and report their locations after every t time units. Our approach can be readily applied to such systems. In the rest of the chapter, we assume a model where a query contacts the server only if it leaves the safe zone.

Although there exists a safe zone based solution for moving window queries [ZZP+03], this technique is not applicable to the moving circular range queries. In Chapter 2, we showed that it is not possible to extend this technique to the case of the distance based range queries as the problems of monitoring moving window queries and the distance based range queries are inherently different. We apply an aggressive approach to prune the objects/entries that cannot affect the results and/or the safe zone. Our pruning rules are tight and the performance of our solution is close to optimal.

We next summarize our contributions.

- We present an efficient and effective technique to monitor the moving circular range queries by adopting the concept of safe zones.
- We present a rigorous theoretical analysis to verify the effectiveness of our safe zone based approach for the moving circular range queries. More specifically, we evaluate the probability that a query moves out of the safe zone within one time unit, the expected distance it travels before it leaves the safe zone, and an upper bound (which is a constant) on the expected number of guard objects for the queries with the diameter of the safe zone no more than a constant times its expected value. Our experimental results confirm the accuracy of the presented theoretical analysis.
- We conduct extensive experiments to show the effectiveness of our approach. We compare our algorithm with an optimal solution and a naïve solution. The experimental results indicate that our proposed approach is close to the optimal solution and an order of magnitude faster than the naïve algorithm.
- Based on non-trivial access order and pruning rules, we present a complete framework for answering the distance based range queries in a road network. Experiments demonstrate that our algorithm is up to two orders of magnitude faster than a naïve algorithm.

The remainder of this chapter is organized as follows. We introduce our framework and pruning rules for processing the moving circular queries in Section 6.2. In Section 6.3, we present our safe zone based solution to the moving circular range queries. Theoretical analysis is presented in Section 6.4. In Section 6.5, we present our techniques to answer moving range queries in a road network. The experimental results are reported in Section 6.6. Section 6.7 summarizes this chapter.

6.2 Framework

6.2.1 Solution Overview

Consider the example in Fig. 6.1 where a range query q is shown. Its range is r and the area within its range is shown shaded. Some objects around it are also shown. The objects that lie within the range form the result set and are called *internal* objects (e.g., the objects o_1 and o_2). The objects that do not lie within the range are called *external* objects (e.g., the object o_3). Let C_i be a circle of radius r with centre at the location of the object o_i. Fig. 6.1 shows the circles for the objects o_1, o_2 and o_3.

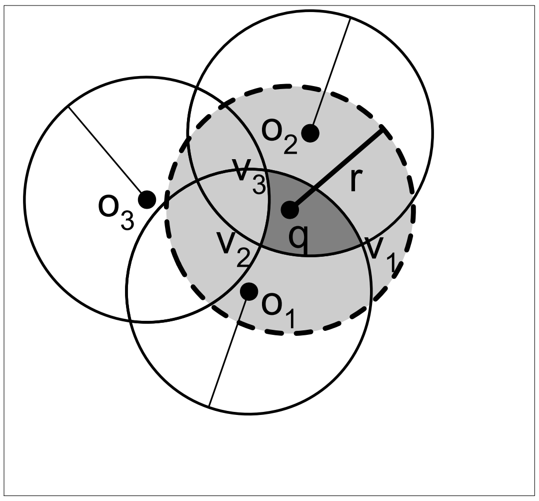


Figure 6.1: A range query (light shaded area) and its safe zone (dark shaded area)

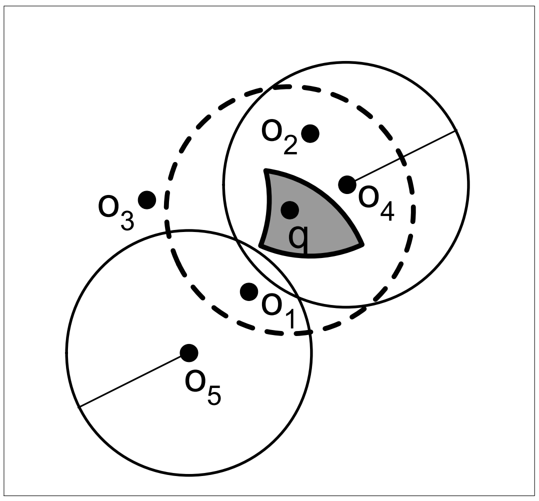


Figure 6.2: Some objects do not affect the safe zone

Note that all the internal objects contain q in their circles whereas the external objects do not. An internal object o_i ceases to be within the range only when the query q leaves its circle C_i. Similarly, an external object becomes included in the result only if the query enters its circle. In other words, the result of the query q does not change as long as q does not leave or enter any circle. Hence, the safe zone of a query q is defined by the boundaries of the circles around it. In the example in Fig. 6.1, the dark shaded area is the safe zone because q does not enter or leave any circle as long as it remains in this area. Formally, safe zone S can be defined as the intersection of the circles of internal objects minus the circles of external objects. That is, $S = \cap_i C_i - \cup_j C_j$ for every internal object o_i and every external object o_j.

Please note that as we consider new objects in order to calculate the safe zone, we may find that some objects may not affect the shape of the safe zone. Consider the example in Fig. 6.2 where the objects o_4 and o_5 are shown. The circle of the internal object o_4 completely contains the current safe zone[1] of q. Hence, it does not change the shape of the current safe zone and will not define the final safe zone. Similarly, the circle of the external object o_5 does not intersect the current safe zone and consequently does not affect its shape. For this reason, the final safe zone can be defined without using the circles of o_4 and o_5. In this chapter, the objects that contribute to the shape of the final safe zone are called *guard* objects (e.g., o_1, o_2 and o_3). An internal (external) object that contributes to the final safe zone is called an internal (external) guard. Internal guards in this example are o_1 and o_2 whereas o_3 is an external guard. For the sake of simplicity, in what follows we refer to both "current safe zone" and "final safe zone" simple as "safe zone".

Data structure at a glance

All objects are indexed by a disk-resident R-Tree [Gut84]. For each query, the server keeps the following information in its memory during the computation of the safe zone: 1) its location; 2) the list of internal objects called *answer list*; 3) the list of guard objects. For each guard object, the server stores its arcs that contribute to the safe zone. In the example in Fig. 6.1, the object o_1 has an arc with two end vertices v_1 and v_3. We use this arc (or vertices) for effective pruning. Note that the server stores this information in its memory only during the construction of the safe zone, and discards this information after the safe zone has been computed and sent to the client.

[1] We use the term current safe zone because the safe zone is being constructed and is not the final safe zone. From now on, the current safe zone is called safe zone and the current guard objects are called guard objects when there is no ambiguity.

Checking whether q lies in the safe zone

Since the clients that issue queries (e.g., mobile devices) have limited computational power, it is desirable that checking whether the client is inside the safe zone is not computationally expensive. Although the shape of a safe zone may be complex, the cost of checking whether q lies in the safe zone takes only k distance computations where k is the number of guard objects. More specifically, the query q computes its distance from each of the guard object. If it lies within the circle of every internal guard and lies outside the circle of every external guard then it lies within the safe zone. Our experimental results show that the average number of guard objects is around 5. We also present a theoretical analysis to give an upper bound on the expected number of guard objects for the queries that satisfy certain constraints.

A simple approach to compute the safe zone is to consider all objects and find the objects that actually contribute to the safe zone. However, the number of objects that are considered must be reduced in order to reduce the I/O cost and to improve the CPU time. We next present five effective pruning rules that significantly reduce the number of considered objects.

6.2.2 Pruning Rules

As shown in the example in Fig. 6.2, some objects do not affect the safe zone. More specifically, if the circle of an object contains the safe zone (such as o_4 in Fig. 6.2) or lies completely outside the safe zone (such as (o_5 in Fig. 6.2), that object does not affect the shape of the safe zone. In this section, we present some effective pruning rules to prune such objects. Note that only the circles of internal objects may contain the safe zone and only the circles of external objects may completely lie outside the safe zone. Hence, some pruning rules are specific to the internal objects and some are to be applied only on external objects.

First, we present pruning rules based on the approximation of the safe zone by a rectangle. Let a and b be two rectangles or points; we use $mindist(a, b)$ and $maxdist(a, b)$

to denote the minimum and maximum distances between them, respectively.

Using approximation of the safe zone

Let R_S be the minimum bounding rectangle of the current safe zone as shown in Fig 6.3. Let R_{cnd} be a rectangle that contains some candidate objects.

Pruning Rule 6.2.1 *If $maxdist(R_{cnd}, R_S) < r$ then no object in R_{cnd} can affect the safe zone.*

Proof Let o be an object in R_{cnd}. For every point $p \in R_S$, $dist(o, p) < r$ because $maxdist(R_{cnd}, R_S) < r$. Hence, the circle of o contains *every* point p of the safe zone, i.e., o does not affect the safe zone. ▮

Pruning Rule 6.2.2 *If $mindist(R_{cnd}, R_S) > r$ then no object in R_{cnd} can affect the safe zone.*

Proof Let o be an object in R_{cnd}. For every point $p \in R_S$, $dist(o, p) > r$ because $mindist(R_{cnd}, R_S) > r$. Hence, the circle of o does not contain *any* point p of the safe zone, i.e., o does not affect the safe zone. ▮

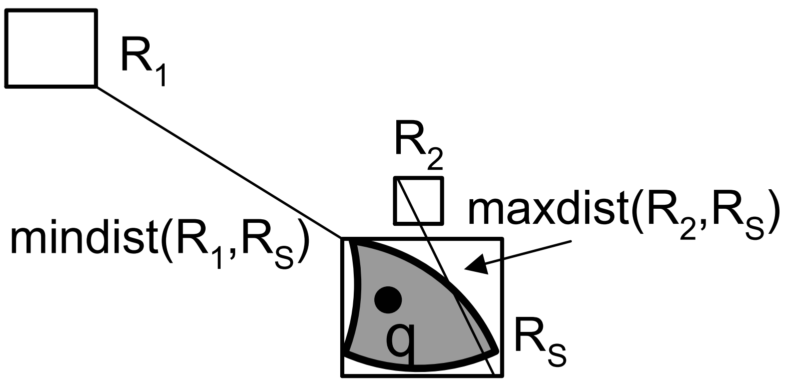


Figure 6.3: Pruning using the approximation of safe zone

In the example of Fig. 6.3, where $maxdist(R_2, R_S) < r$, it can be immediately verified that any object in R_2 contains the safe zone in its circle. Similarly, $mindist(R_1, R_S) > r$ and every object in R_1 can also be pruned.

Using the guard objects

Although the rectangle based pruning is inexpensive, it is unfortunately not very tight. We present tighter pruning rules below, based on the positions of the guard objects.

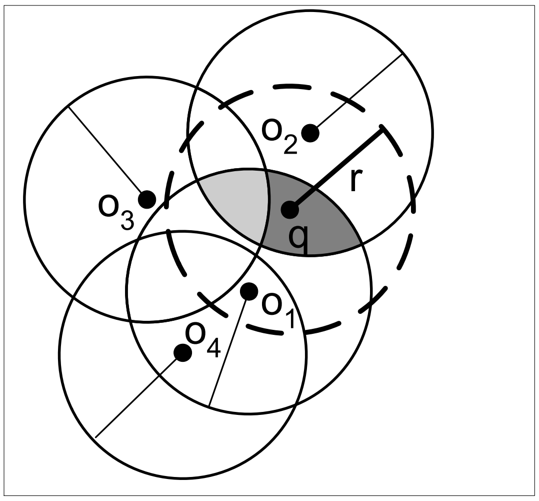


Figure 6.4: Illustration of pruning rule 6.2.3

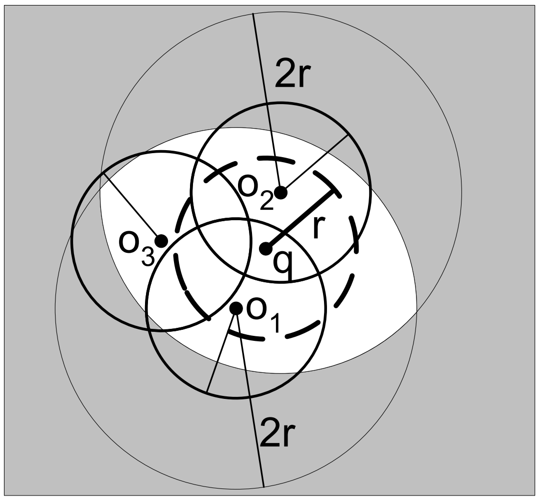


Figure 6.5: Area pruned by the rule 6.2.3

Pruning Rule 6.2.3 *If* $mindist(R_{cnd}, o_i) > 2r$ *for* any *internal guard object* o_i *then no object in* R_{cnd} *can affect the safe zone.*

Proof An object can only affect the safe zone if its circle intersects the safe zone. Safe zone is the area defined by the intersection of the circles of the internal guard objects minus the circles of the external guard objects. Hence, the circle of any internal guard object contains the whole safe zone, Thus a circle can only intersect the safe zone if it intersects the circles of *all* internal guard objects. Consequently, if an object o_j lies at a distance greater than $2r$ from any internal guard o_i, it cannot intersect the safe zone. ∎

In Fig. 6.4, the object o_4 cannot affect the safe zone because it lies at a distance greater than $2r$ from o_2. To show the area that is pruned by this pruning rule, we zoom out Fig. 6.4 and show the pruned area in Fig. 6.5. The shaded area can be pruned because every point in it lies at a distance greater than $2r$ from at least one of o_1 and o_2. This pruning rule prunes the rectangles that contain external objects.

Before we present tighter pruning rules, we provide few auxiliary observations and lemmas.

Consider a circle C with centre at M and radius r, and any point E in the plane (inside or outside the circle) (see Fig. 6.6). The line that passes through E and M intersects the circle at two points, A and B. Without loss of generality, we assume that $dist(A, E) < dist(B, E)$, as shown in Fig. 6.6. We make the following observation.

Observation 6.2.4 *Let C be a circle of radius r, and M, E, A and B be the points as described above. The distance between E and any point D on the circle monotonically increases as D moves along the circle from point A to B, either clockwise or counter-clockwise. In other words, any point D' that lies before D while travelling on the circle from A to B satisfies $dist(E, D') < dist(E, D)$.*

The above observation can be easily verified from the triangle $\triangle EMD$. If we denote $\overline{MD}$ by r and the length of $\overline{EM}$ by x, then the length of $\overline{DE}$ is given by the law of cosine as $dist(D, E) = \sqrt{r^2 + x^2 - 2rx \cdot cos(\angle EMD)}$. Note that as D travels along the circle from A to B, the angle $\angle EMD$ increases from $0°$ to $180°$ and its cosine monotonically decreases from 1 to -1. As both r and x remain unchanged, the distance $dist(D, E)$ monotonically increases. Note that we do not require x to be smaller than r, so the observation also holds for the case when E lies outside the circle.

Based on Observation 6.2.4, we present the following lemma that is used in our next pruning rule.

Lemma 6.2.5 *Let $\widehat{AB}$ be an arc of radius r with subtending angle $\theta < 180°$ where A and B are the end points of the arc and M is the centre (as shown in Fig. 6.7). Let C_A and C_B be two circles of radius r centred at A and B, respectively. Every point E that lies inside both the circle C_A and circle C_B satisfies the following: The circle of radius r with centre at E (the dotted circle in Fig. 6.7) contains every point of the arc $\widehat{AB}$.*

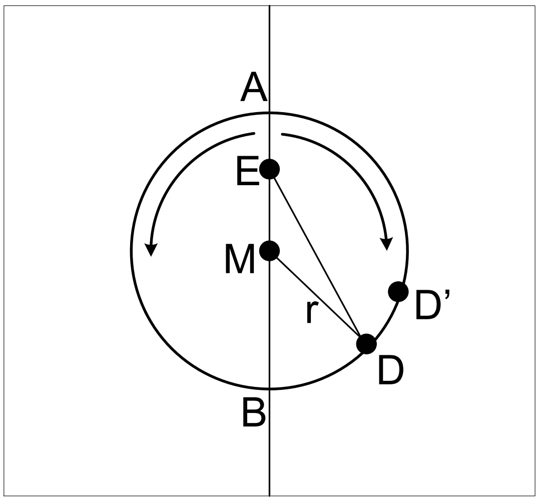


Figure 6.6: Observation 6.2.4

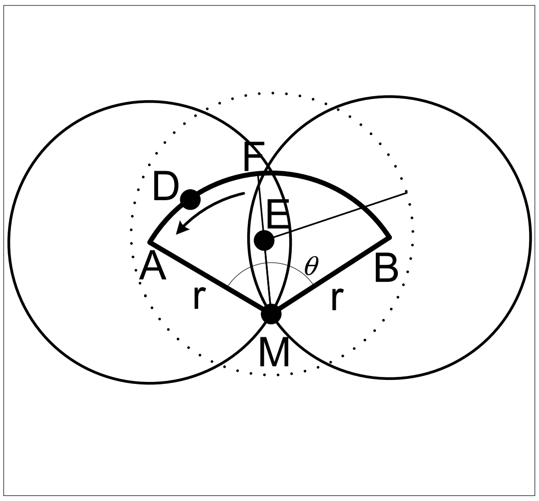


Figure 6.7: Lemma 6.2.5

Proof In order to prove the lemma, we need to show that the distance of E from any point D that lies on the arc $\widehat{AB}$ is smaller than r. If we extend the line joining M and E, it cuts the arc at point F which is the minimum distance from E to the circle. We prove the lemma for the arc $\widehat{AF}$ and the proof for the arc $\widehat{FB}$ is similar. By Observation 6.2.4, we know that any point D that lies on the arc $\widehat{AF}$ satisfies $dist(E, D) \leq dist(E, A)$. As the point E lies inside the circle C_A, $dist(E, A) < r$. Hence, $dist(E, D) < r$ for any point D. ∎

Please note that the lemma does not hold if the subtending angle $\theta \geq 180°$ as the line joining M and E intersects the arc $\widehat{AB}$ at point F which is the maximum distance from E to the circle and is greater than r (Fig. 6.8).

Based on Lemma 6.2.5, we present a pruning rule to prune the rectangles that contain internal objects.

Pruning Rule 6.2.6 *Let S be a safe zone such that every arc that defines it has subtending angle smaller than 180°. If $maxdist(R_{cnd}, v_i) \leq r$ for every vertex v_i of the safe zone S, then no object in R_{cnd} can affect the shape of the safe zone.*

Proof Let E be a point that lies within all the circles of radius r centred at vertices of the safe zone. From Lemma 6.2.5, we know that the circle centred at E contains every

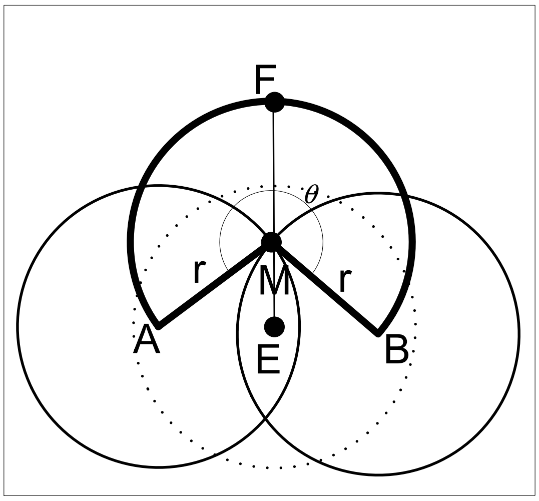


Figure 6.8: When $\theta > 180°$

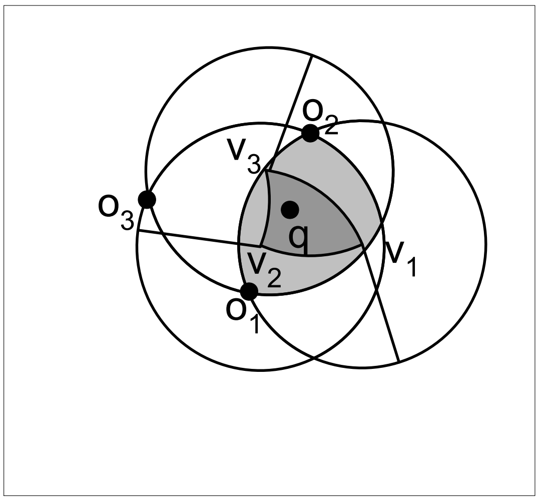


Figure 6.9: Pruning rule 6.2.6

arc of the safe zone. Hence, it contains the whole safe zone and cannot affect its shape. ∎

Fig. 6.9 shows three circles of radius r with centres at the vertices v_1, v_2 and v_3. Any object or rectangle that lies in the shaded area can be pruned because its distance to any vertex cannot be greater than r.

For our final pruning rule, we need the following lemma.

Lemma 6.2.7 *Let $\overset{\frown}{AB}$ be an arc with centre at M, radius r and subtending angle $0 < \theta < 360°$ as shown in Fig. 6.10. The distance of E from every point of the arc $\overset{\frown}{AB}$ is greater than r, if E satisfies either of the following conditions:*

1) E lies within the angle range θ and $dist(E, M) > 2r$;

2) E lies outside the angle range θ, $dist(E, A) > r$ and $dist(E, B) > r$.

Less formally, if E lies within the shaded area in Fig. 6.10, its distance to any point on the arc $\overset{\frown}{AB}$ is greater than r.

Proof We first consider a point E_1 that lies within the angle range θ (see Fig. 6.10). We draw a line through points E_1 and M and we denote the intersection of the line and the arc by G. By Observation 6.2.4 $dist(E_1, G)$ is the minimum distance from the point E to the arc $\widehat{AB}$. Since $dist(E_1, M) > 2r$, it follows that $dist(E_1, G) > r$ and thus $dist(E_1, D) > r$ for any point D on the arc $\widehat{AB}$.

We now consider a point E_2 that lies outside the angle range θ (see Fig. 6.10). Again, by Observation 6.2.4, the minimum distance from E_2 to the circle is $dist(E_2, F)$ (see Fig. 6.10), and the distance between E_2 and the points on the circle increases monotonically as we move along the circle away from the point F. Thus for every point D on the arc $\widehat{AB}$ we have either $dist(E_2, D) \geq dist(E_2, A) > r$ or $dist(E_2, D) \geq dist(E_2, B) > r$

■

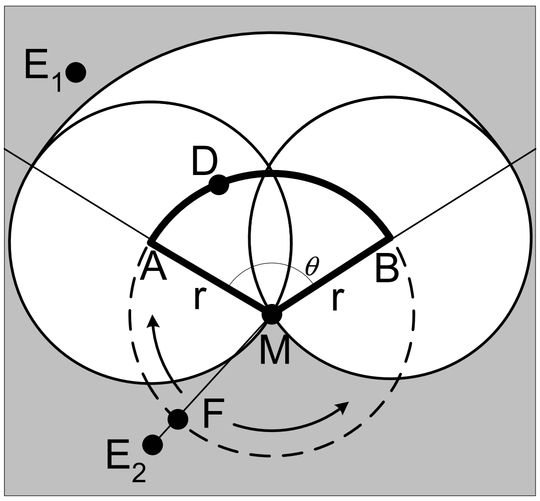


Figure 6.10: Illustration of Lemma 6.2.7

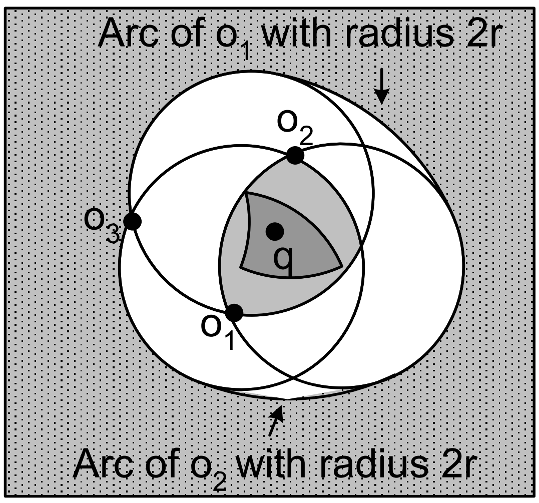


Figure 6.11: Pruning rules 6.2.6 and 6.2.8

Based on Lemma 6.2.7, we present our final pruning rule that prunes external objects.

Pruning Rule 6.2.8 *No object in a rectangle R_{cnd} can affect the safe zone if R_{cnd} satisfies Lemma 6.2.7 (i.e., R_{cnd} lies completely in the shaded area of Fig. 6.10) for every arc of the safe zone.*

Proof The proof immediately follows from Lemma 6.2.7 as any point in R_{cnd} has minimum distance to the boundary of the safe zone greater than r. Hence, its circle cannot intersect the safe zone. ■

In order to apply this pruning rule, we check the minimum distance of the rectangle R_{cnd} from M, A and B. If the rectangle completely lies outside the angle range θ, it can be pruned if its minimum distance from both A and B is greater than r. Otherwise, it can be pruned if its minimum distance from M is greater than $2r$.

Fig. 6.11 shows the area pruned by the rules 6.2.6 and 6.2.8, where the outer shaded area is pruned by the pruning rule 6.2.8 and we call it *external* pruned area. The inner shaded area is pruned by the rule 6.2.6 and we call it *internal* pruned area.

The arguments similar to those used in proofs of Lemma 6.2.5 and 6.2.7 can be used to show that the pruning rules are tight. In other words, any object that lies in the unpruned area (the white area in Fig. 6.11) affects the shape of the current safe zone. Note that although the rectangle based pruning rules have less pruning power, they are important because they are computationally less expensive. We first apply the rectangle based pruning rules and if an object is not pruned, we apply the guard objects based pruning rules.

6.3 Technique

Initially, the whole space is assumed to be the safe zone. We then access each object that cannot be pruned, and use its circle to trim the safe zone. The algorithm stops when all the objects that cannot be pruned are accessed. The order in which the objects are accessed is important as better access order retrieves fewer objects that affect the safe zone. We first present our proposed access order. Secondly, we present our query processing algorithm followed by the algorithm to trim the safe zone. Finally, we present an efficient technique to update the safe zone when the query leaves it.

6.3.1 Access order

After applying the pruning rules presented above, there may be several objects left in the unpruned area. The order in which these objects are accessed is important. Intuitively, the objects that lie closer to the boundary of the range query have a more significant effect on the shape of the safe zone and should be accessed first.

Consider the example in Fig. 6.12, where the boundary of q is shown in thick broken line. The objects o_1, o_2 and o_3 are accessed first and are the current guard objects. The object o_4 that lies closer to the boundary than all of the existing guard objects is guaranteed to affect the shape of the safe zone. In Fig. 6.13, the object o_4 is accessed and the safe zone is shown after trimming with respect to its circle. We present a lemma that shows the importance of the objects located near the boundary for constructing the safe zone.

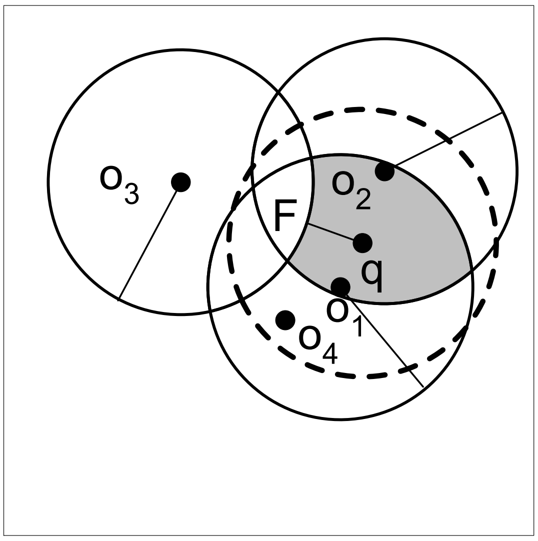


Figure 6.12: Importance of access order

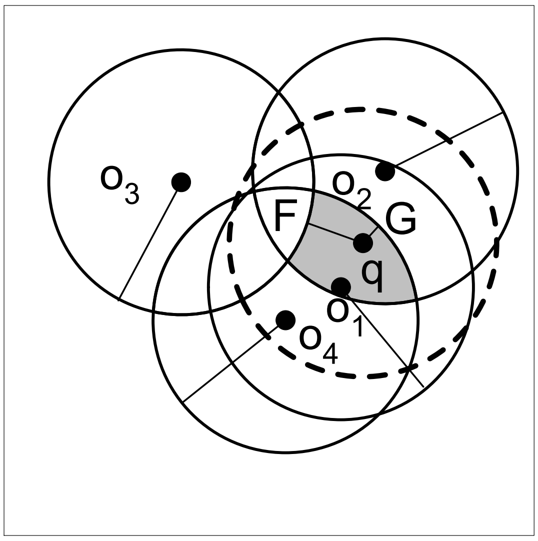


Figure 6.13: o_1 is no more a guard object

Lemma 6.3.1 *Let o_i be an object that is closer to the boundary of the range query than all current guard objects. The object o_i is guaranteed to affect the shape of the current safe zone.*

Proof Without loss of generality, consider the example in Fig. 6.12 where the current safe zone is shown shaded. The closest guard object to the boundary of the range query is o_3. Thus the minimum distance from the query to the current safe zone is $\mid dist(o_3, q) - r \mid$. Any object o_4 that lies closer to the boundary than o_3 has a point G on its circle with distance $\mid dist(o_4, q) - r \mid$ from the query, which is less than $\mid dist(o_3, q) - r \mid$ (see Fig. 6.13).

Hence, the circle of o_4 has at least one point inside the current safe zone so it affects the safe zone. ∎

In fact, in this particular example, the object o_4 is not only a guard object but it also removes the object o_1 from the list of the guard objects. Consider Fig. 6.13, where the object o_4 has been considered for trimming and the new safe zone is shown shaded after. Clearly, the circle of the object o_1 does not contribute to the safe zone anymore, and consequently o_1 is removed from the list of the guard objects. This example supports the intuition that the objects that lie closer to the boundary of the query should be accessed first. Our experimental results demonstrate the effectiveness of this proposed access order (Fig. 6.29 in Section 6.6). Next, we present an efficient algorithm that accesses the objects in the proposed order.

6.3.2 Algorithm

We use an R-Tree [Gut84] to index the objects. Each leaf and index node of an R-tree contains pointers to its entries and a minimum bounding rectangle that contains all its objects. For details, please see [Gut84].

Algorithm 12 outlines the solution. A min-heap is initialized with the root entry of the R-tree. The entries are de-heaped iteratively until the heap becomes empty. If a de-heaped entry e has $maxdist(e, q) < r$, then all the objects in it are internal and we apply pruning rules 6.2.1 and 6.2.6. If the entry is pruned, we do not need to check any objects within it for the construction of the safe zone. However, as these objects are internal, they contribute to the answer to be sent to the query. Therefore, we insert all the objects that are within this entry to the answer list (lines 4 - 7).

If the de-heaped entry e has $mindist(e, q) > r$, then all the objects in it are external objects and we apply pruning rules 6.2.2, 6.2.3 and 6.2.8 (lines 8 and 9). If the entry is pruned, we continue the algorithm by de-heaping the next entry. Note that an entry e for which $mindist(e, q) \leq r \leq maxdist(e, q)$ cannot be pruned by any of the pruning rules. This is because such entries may contain both internal and external objects, while all the

Algorithm 12 Range Query (q, r)

Input: q: the query point; r: range of the query;

Description:

```
1: initialize a min-heap H with root of the R-Tree
2: while H is not empty do
3:   deheap an entry $e$
4:   if $maxdist(e, q) < r$ then
5:     if pruned using rules 6.2.1 and 6.2.6 then
6:       insert all objects of $e$ in the answer list
7:       continue
8:   else if $mindist(e, q) > r)$ then
9:     If pruned using rules 6.2.2, 6.2.3 and 6.2.8, continue;
10:  if $e$ is an object then
11:    TrimSafeZone($e$,$q$,$S$)  /* Algorithm 13 */
12:    if $e$ is an internal object, insert in the answer list
13:  if $e$ is a leaf or index node then
14:    for each entry c in $e$ do
15:      insert $c$ into H with key set to its minimum distance from boundary
16: send guard objects and answer list to the query $q$
```

proposed pruning rules are applicable either to internal objects or to external objects. For this reason, we do not consider such entries for pruning.

If e is an object and cannot be pruned, we use it to trim the safe zone; if it is an internal object, we also insert it into the answer list (lines 10 - 12). Otherwise, if e is a leaf or index node, we insert its entries into the heap with key of each entry set to minimum distance of the entry from the boundary of the range query (lines 13 - 15). The algorithm stops when the heap becomes empty.

The minimum distance of an entry e from the boundary of the range query is computed as follows: If $mindist(e, q) \leq r$ and $maxdist(e, q) \geq r$, then the minimum distance of this entry from the boundary is zero because the entry e overlaps the boundary (see R_1 in Fig. 6.14). If $mindist(e,q) > r$, then the minimum distance of this entry is $mindist(e,q)-r$ (see R_2 in Fig. 6.14). Finally, if the $maxdist(e,q) < r$ then the minimum distance is $r - maxdist(e,q)$ (see R_3 in Fig. 6.14).

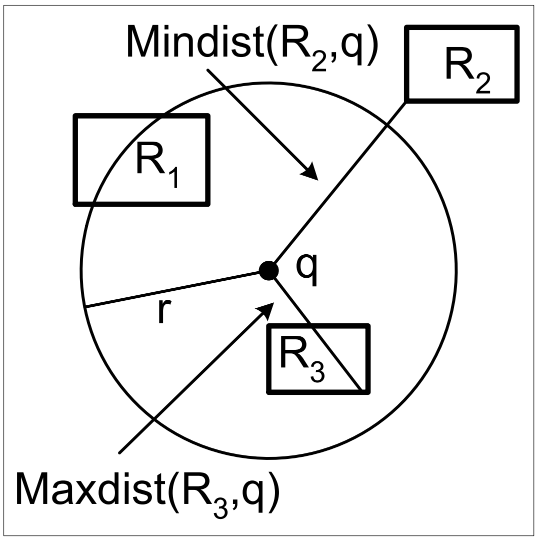


Figure 6.14: Minimum distance from the boundary

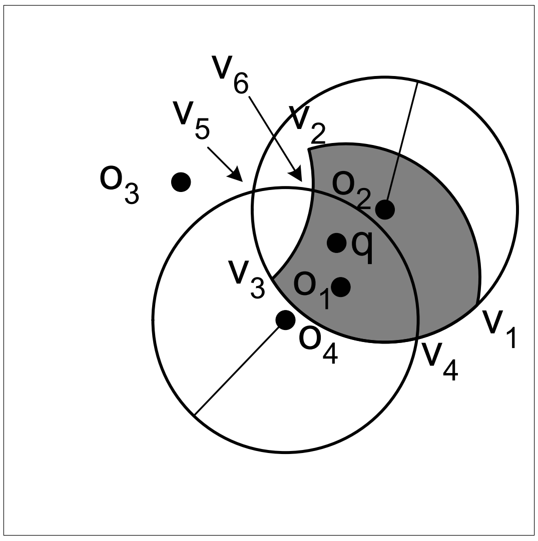


Figure 6.15: Illustration of the trimming (Algorithm 13)

In a special case when there is no object within the range, the whole space minus the circles of all the external objects will be the safe zone. However, the number of guard objects may be arbitrarily large. For such cases, in order to restrict the space, we treat query location as a virtual internal object. Then only the objects within distance $2r$ of the query may be the guard objects.

6.3.3 Trimming the safe zone

Algorithm 13 shows the procedure to trim the safe zone with respect to an object o. Note that to trim the safe zone, we only need to update the guard objects and the vertices of the safe zone and we do it as follows. For each guard object o_i, the intersection points of the circles of o and o_i are computed. If the intersection point lies on the boundary of the safe zone, the point is added as the vertex of the safe zone (lines 1 to 3). Then, the object o is added as the guard object.

Algorithm 13 TrimSafeZone (o, q, S)

Input: o: an object o to be used for updating the safe zone; q: the query point; S: the list of current guard objects;

Description:

```
1: for each guard object $o_i$ in $S$ do
2:     for each intersection point $v_i$ of circles of $o$ and $o_i$ do
3:         add $v_i$ to vertices list if $v_i$ lies on the boundary of the safe zone
4: add $o$ to the list of guard objects $S$
5: if $o$ is an internal object then
6:     remove every vertex $v$ if $dist(o, v) > r$
7: else if $o$ is an external object then
8:     remove every vertex $v$ if $dist(o, v) < r$
9: remove every guard object $o$ from $S$ if all its related vertices have been removed
```

Finally, the existing vertices that are no longer in the safe zone are removed and the objects that no longer have any associated vertices are removed from the list of guard objects (lines 5 to 9).

Fig. 6.15 illustrates the Algorithm 13 and shows the safe zone (shaded), together with its current guard objects o_1, o_2 and o_3. The safe zone is to be trimmed by a new object o_4. For the sake of clarity, the circles of o_1 and o_3 are not shown. The circle C_4 of the object o_4 intersects the circle C_2 of the object o_2 at two points, v_4 and v_5. The intersection point v_4 lies on the boundary of safe zone, so it is added to the list of vertices of the current safe zone. The intersection point v_5 lies outside the safe zone so it is deleted. Similarly, the intersection points of the circle C_4 with the circles of o_1 and o_3 are considered and v_6 is added to the list of vertices. All other intersection points lie outside the safe zone and are deleted.

Now the vertices of the safe zone that are not valid anymore are to be deleted. Since o_4 is an internal object (it contains q in its circle), all vertices that lie outside its circle are deleted. For this reason, the vertices v_1 and v_2 are deleted. The related object o_1 is also deleted as it no longer has any associated vertex. After trimming of the safe zone, its vertices are v_3, v_4 and v_6 and the guard objects are o_2, o_3 and o_4.

6.3.4 Updating the safe zone when query leaves it

When the query leaves its safe zone, it sends its current location and current guard objects to the server. The server updates the answer list (the list of internal objects), computes the new safe zone and sends it to the query. A straightforward approach is to compute the safe zone and answer list from scratch. However, this is not only expensive but can also cause a large amount of data to be transmitted from the server to the query if the answer list contains a large number of objects.

In this section, we propose an effective approach to update the safe zone and the answer list, called *smart-update*. The smart-update utilizes the previous safe zone of the query and avoids searching the area that was visited before. Furthermore, instead of computing and sending all the objects lying within the range, the smart-update sends a list of objects called *delta list* that contains two types of objects. An object o_i^+ indicates that the object o_i that was previously external is now internal. So, the client must add it in its answer list. An object o_i^- indicates that the object o_i that was previously internal is now external. Hence, the client must remove it from its answer list.

Fig. 6.16 shows that a query q leaves the safe zone and moves to q'. The shaded area corresponds to the area that was pruned with respect to its previous safe zone. The smart-updates first considers the existing guard objects and constructs an initial safe zone (as shown in Fig. 6.17). Then, the smart-update uses two observations to reduce the search area. 1) The white area of the Fig. 6.16 cannot contain any object. The proof is straightforward because if there were any object in the white area, it would have affected the previous safe zone. Hence, the smart-update does not search this area. 2) The query q contains in its answer list all the objects that are in the internal pruned area (the internal shaded area of Fig. 6.16). Hence, the objects that lie within distance r from

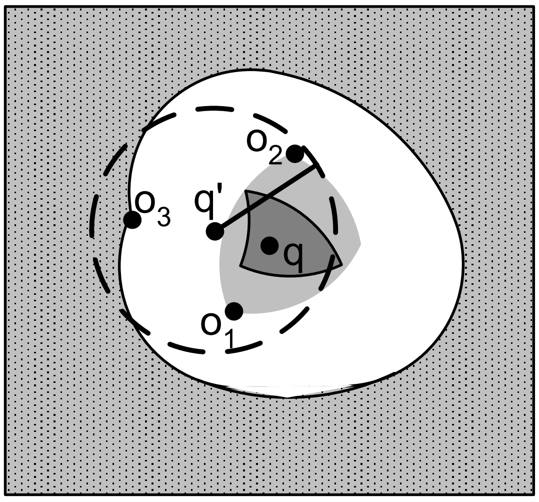


Figure 6.16: q leaves the safe zone

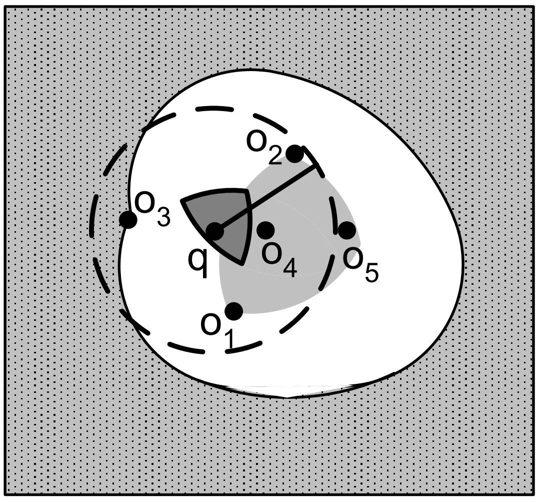


Figure 6.17: Smart-update in action

q' and lie in the internal pruned area are not required to be sent to the client.

In the example in Fig. 6.17, the object o_4 is not sent to the query because it lies in the previous internal pruned area and the query already contains it. However, the object o_5 must be sent so that the query removes it from its answer list.

6.4 Theoretical Analysis

In this section we present a theoretical analysis to evaluate the effectiveness of the safe zone. In what follows we assume that there are N objects in total and that they are uniformly distributed in a square unit universe.

6.4.1 Escape Probability (P_{esc})

We first analyse the *escape probability* P_{esc}, which we define as the probability that a query q leaves its safe zone within one time unit. Escape probability is important because a smaller escape probability indicates that on average the results of the query will remain unchanged for longer.

Consider the example in Fig. 6.18 with a range query q and the guard objects o_1, o_2 and o_3. The safe zone is shown with bold boundary. Suppose that the query q travels some distance x along a straight line in an arbitrary direction and that it crosses the

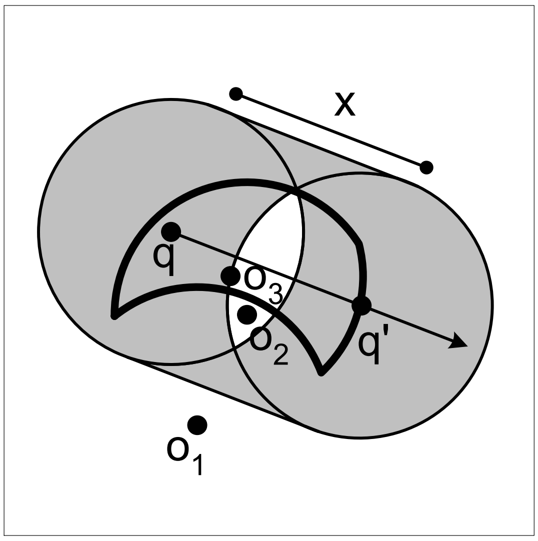


Figure 6.18: Sweeping region ($x < 2r$)

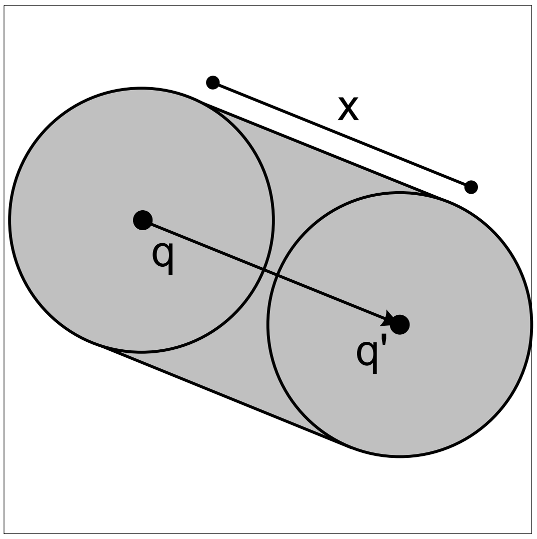


Figure 6.19: Sweeping region ($x \geq 2r$)

boundary of the safe zone at point q'. Zhang et al. [ZZP+03] presented an interesting observation for window queries which we here apply to the circular range queries. When a query q moves, its circle sweeps some area, which is called *sweeping region*. In Fig. 6.18, the shaded area corresponds to the sweeping region of the query which moved from q to q'. It is important to note that as long as the query remains in the safe zone, that is, while $x \leq dist(q, q')$, the corresponding sweeping region contains no objects.

The area A of the sweeping region when the query moves a distance $x < 2r$ (as shown in Fig. 6.18) and a distance $x \geq 2r$ (as shown in Fig. 6.19) is

$$A(x) = \pi r^2 + 2rx - \begin{cases} 2r^2 arccos(\frac{x}{2r}) - x\sqrt{r^2 - \frac{x^2}{4}}\,, \text{ if } x < 2r \\ 0\,, \text{ otherwise} \end{cases} \tag{6.1}$$

Since we assume uniform distribution of the objects in a unit universe, the probability p_i that an object o_i lies within the sweeping region is $A(x)$. The probability p'_i that the object o_i does not lie within the sweeping region is $(1 - A(x))$. The probability that none of the N objects lies within the sweeping region is $(1 - A(x))^N$. Hence, the probability that the query does not leave its safe zone when traveling a distance x, i.e., the probability that $x < dist(q, q')$ is $(1 - A(x))^N$. Finally, the probability that at least one of the N

objects lies within the sweeping region, that is, the probability that $x \geq dist(q, q')$ is:

$$P\{x \geq dist(q, q')\} = 1 - (1 - A(x))^N \tag{6.2}$$

Let the query speed v be such that the query travels distance d in one time unit. The probability of escape P_{esc} can be computed as $P\{d \geq dist(q, q')\} = 1 - (1 - A(d))^N$.

6.4.2 Expected distance (m)

In this section, we analyse the expected distance m that a query travels before it leaves its safe zone. The probability density function $pdf(x)$ is given by the derivative of $P(x)$ presented in Equation (6.2) as follows:

$$pdf(x) = 2rN(1 - A(x))^{N-1} \begin{cases} (1 + \sqrt{1 - (\frac{x}{2r})^2})\,, \text{ if } x < 2r \\ 1\,, \text{ otherwise} \end{cases} \tag{6.3}$$

Integrating $x \cdot pdf(x)dx$ for x from 0 to 1 gives us the expected distance.

Unfortunately, it is difficult to integrate $x \cdot pdf(x)dx$ because the area A is represented by trigonometric functions and it makes the expression difficult to solve when $x < 2r$. We address this problem by approximating the area $A(x)$ when $x < 2r$. By plotting the equations on a graph, it can be shown that when $0 \leq x \leq 2r$, then $1.1\pi rx \leq A(x) \leq 1.3\pi rx$. We thus define the lower bound on the area as $A_{low} = 1.1\pi rx$ and the upper bound as $A_{up} = 1.3\pi rx$. We can then show that for $x < 2r$, $2rNx(1 - A_{up})^{N-1} \leq x \cdot pdf(x)dx \leq 4RNx(1 - A_{low})^{N-1}$. Thus we define the lower and upper bound on the expected distance as follows:

$$m_{up} = \int_0^{2r} 4rNx(1 - A_{low})^{N-1}dx + \int_{2r}^{1} 2rNx(1 - A(x))^{N-1}dx \tag{6.4}$$

$$m_{low} = \int_0^{2r} 2rNx(1 - A_{up})^{N-1}dx + \int_{2r}^{1} 2rNx(1 - A(x))^{N-1}dx \tag{6.5}$$

Exact values of m_{low} and m_{up} can be found by solving the integrals. For large values of N we have

$$m_{up} \approx \frac{0.33}{rN} \quad \text{and} \quad m_{low} \approx \frac{0.12}{rN} \tag{6.6}$$

The equations for the expected distance bounds describe the relation between the expected distances, radius and the total number of objects. More specifically, the expected distance is inversely proportional to the radius r and the number of objects N.

6.4.3 Expected number of guard objects

We now evaluate the expected number G of guard objects. Let $d(\theta)$ be the distance a query moves in direction θ before it leaves the safe zone. Let d_{max} be the maximum of $d(\theta)$ over all θ such that $0 \leq \theta \leq 2\pi$. Let $P(x)$ be the probability that a query has $d_{max} \leq x$. We know from the theory of conditional expectation that the expected number of guard objects is given by

$$E(G) = \int_{o}^{1} E(G|d_{max} = x)P'(x)dx \tag{6.7}$$

where $E(G|d_{max} = x)$ is the expected number of guard objects for a query that has $d_{max} = x$ and $P'(x)$ is the derivative of $P(x)$ with respect to x. First, we show that $E(G|d_{max} = x) \leq 4\pi r x N$.

Consider the example of Fig. 6.20 where the maximum distance from q to the boundary of the safe zone is x (x corresponds to the circle shown in thick line). The circles of radii r, $r + x$ and $r - x$ are also shown. Any object o_i that lies in the circle of radius $r - x$ cannot be a guard object because the circle C_i of the object o_i fully contains the safe zone. This is the case because the maximum distance of o_i to the safe zone $maxdist(o_i, S) \leq dist(q, o_i) + x \leq r$. Hence, the object o_i cannot affect the shape of the safe zone.

Similarly, any object o_j that lies outside the circle of radius $r + x$ cannot affect the shape of the safe zone as the minimum distance of o_j to the safe zone $mindist(o_j, S) \geq$

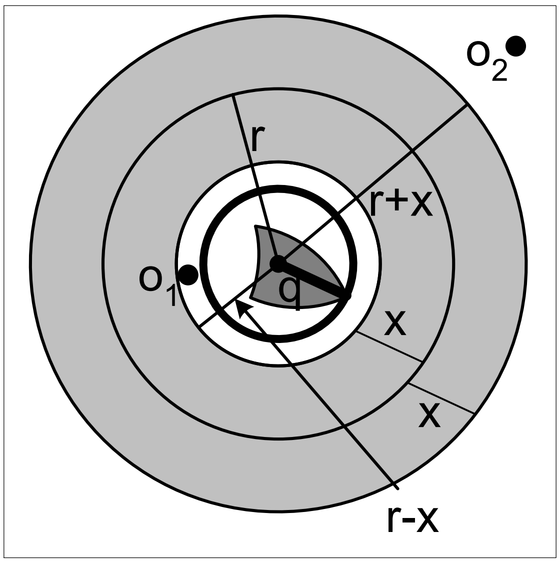


Figure 6.20: Proving that $E(G|d_{max} = x) < 4\pi rxN$

$dist(q, o_j) - x \geq r$. Fig. 6.20 shows two objects o_1 and o_2 and both cannot be the guard objects.

As discussed above, only those objects that have a distance from the query no less than $r - x$ and no greater than $r + x$ can be the guard objects (i.e., only the objects in the area shown shaded in Fig. 6.20 can be the guard objects). Thus the number G of guard objects of any query with $d_{max} \leq x$ is less than or equal to the total number of objects in the shaded area and consequently the expected number of G is less than or equal to the expected number of objects in the shaded area which is $(\pi(r+x)^2 - \pi(r-x)^2)N = 4\pi rxN$. Hence $E(G|d_{max} = x) \leq 4\pi rxN$.

For queries q for which $d_{max} \leq C \cdot m$, where C is a constant and m is the expected distance, Equation (6.8) shows the upper bound of the expected number of guard objects. In other words, if we consider only the queries for which the maximum distance to the boundary of the safe zone d_{max} is not greater than $C \cdot m$, the upper bound on the expected number of guard objects is given by

$$\begin{aligned}\int_0^{C\cdot m} E(G|d_{max}=x)P'(x)dx \leq 4\pi rNCm\int_o^{C\cdot m} P'(x)dx \\ = C\cdot 4\pi rmN\end{aligned} \tag{6.8}$$

Hence, the queries that have $d_{max} \leq C\cdot m_{up}$ have the expected number of guard objects at most:

$$4\pi rNC \times \frac{0.33}{rN} = 4.14C \tag{6.9}$$

If we know C, we can obtain the upper bound on the expected number of guard objects. For instance, in our experiments, we found that 30% to 50% of the queries have d_{max} less than $2m_{up}$ (i.e., C is at most 2). Hence, the upper bound for such queries is 8.28.

6.5 Range Queries in Road Networks

6.5.1 Solution Overview

Before we outline our approach, we define a few terms.

A road network G is a weighted graph consisting of *vertices* and *edges*. An edge between two vertices v_1 and v_2 is denoted as $e(v_1, v_2)$. Each edge has a positive weight that denotes the cost of travelling on that edge (e.g., length of the edge, time taken to travel along the edge etc.).

Fig. 6.21 shows an example of a road network, together with three objects (o_1, o_2 and o_3) and a query q. For simplicity, the objects o_1, o_2 and the query q are chosen to coincide with vertices of the graph.

Segment $seg(x, y)$ is the part of an edge between x and y where both x and y are points on the edge. By definition, an edge is also a segment defined by the end points (vertices) of the edge. Fig. 6.21 shows several segments including segment $seg(b, h)$ of weight 7.

Minimum network distance $MinNetDist(a, b)$ between any two points a and b is the minimum distance from a to b (i.e., total weight of the edges and segments on the shortest path from a to b). For example, the shortest path between o_3 and v is $o_3 \rightarrow b \rightarrow a \rightarrow v$ and the $MinNetDist(o_3, v)$ is 14.

Range network of a point p (denoted as RN_p) for a given range r consists of every point of the road network G that is within the network distance r from the point p. Fig. 6.21 shows the range network ($r = 10$) of o_3 in thick lines.

Internal/external objects and vertices. All objects (vertices) that lie on the range network of the query q are called internal objects (vertices) and all other objects (vertices) are called external objects (vertices). Although the range network of q is not highlighted in Fig. 6.21, it is easy to see that the objects o_1 and o_2 are internal objects and o_3 is an external object. The vertices q, t, s, w, a are internal vertices and all other vertices are external vertices.

Safe zone is a *connected* network consisting of edges and segments such that as long as the query remains in the safe zone, its result does not change. In the example of Fig. 6.21, the safe zone is shown with broken lines. More specifically, the safe zone consists of $e(q, w), e(q, t), e(q, s), e(s, t)$ and $seg(s, v)$. Please note that as long as the query remains on these edges and segments the results remain the same.

The main idea of our solution is similar to our safe zone based approach for Euclidean space. More specifically, the safe zone in a road network consists of the segments of the network that are within distance r from each internal object and have distance greater than r from each external object. In other words, the safe zone is the intersection of range networks of all internal objects minus range networks of all external objects. Formally, the safe zone S is given by the expression $S = \cap_i RN_i - \cup_x RN_x$, where the intersection is taken over the range networks of all internal objects and the union is taken over the range networks of all external object.

Checking whether q lies in the safe zone

In contrast to the safe zone of circular range queries, the safe zone in a road network consists of edges and segments. The safe zone (e.g., the edges and segments) is sent to the query and it can easily check whether it lies in the safe zone or not.

6.5.2 Pruning Rules

Pruning internal objects

Pruning Rule 6.5.1 *An internal object i cannot affect the safe zone if its range network RN_i contains the whole safe zone.*

Proof Recall that the safe zone is given by $S = \cap_i RN_i - \cup_x RN_x$. If the range network of an internal object i contains the whole safe zone, it implies that the intersection of the current safe zone and the range network of i is the same as the current safe zone. Hence, the safe zone is not affected. ∎

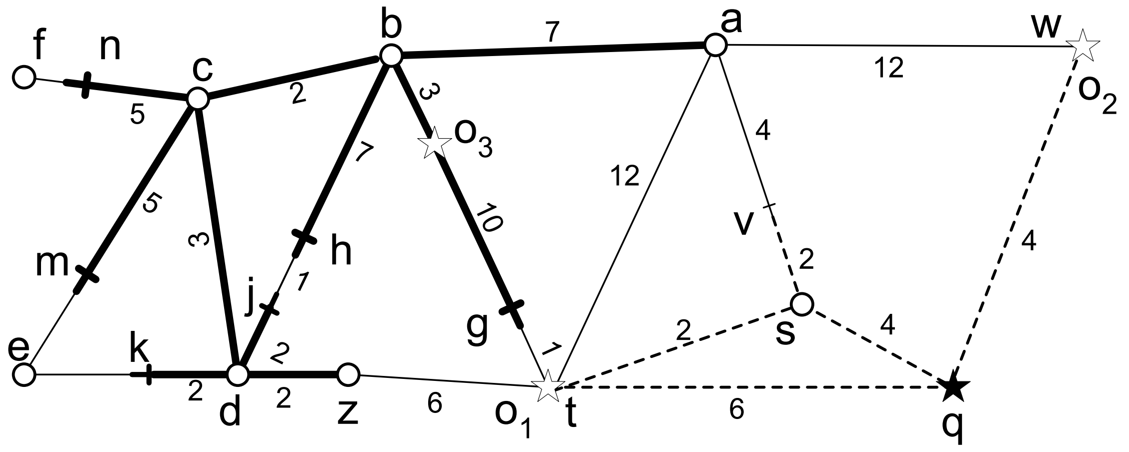


Figure 6.21: Range query on a road network ($r = 10$)

Consider the example of Fig. 6.21 and assume that there is an object o_4 (not shown in the figure) that lies anywhere on the edge $e(q, t)$. Such an object would not affect the safe zone because its range network would cover the whole safe zone.

The above pruning rule requires computing the range networks of the internal objects in order to prune them. Next, we present a pruning rule that is less expensive.

Pruning Rule 6.5.2 *Let d_{max} be the maximum $MinNetDist(q,x)$ over every point x in the safe zone (i.e., $d_{max} = max_{x \in S}(MinNetDist(q,x))$ where S denotes the safe zone). An object o such that $MinNetDist(o,q) \leq (r - d_{max})$ cannot affect the safe zone.*

Proof We prove this by showing that the range network of any such object o contains the whole safe zone. Let x be any point in the safe zone S. The network distance between x and o satisfies $MinNetDist(o,x) \leq MinNetDist(o,q) + MinNetDist(q,x)$. Since $MinNetDist(o,q) \leq (r - d_{max})$ and $MinNetDist(q,x) \leq d_{max}$, $MinNetDist(o,x) \leq (r - d_{max}) + d_{max} \leq r$. Hence, the range network of the object o contains every point x of the safe zone. ■

In the example of Fig. 6.21, d_{max} is 6. Hence, any object that lies within distance $10 - d_{max} = 4$ of q cannot affect the safe zone. To prune an internal object, we first apply the pruning rule 6.5.2 (due to its low cost) and then apply pruning rule 6.5.1.

Pruning external objects

We next present the pruning rules for external objects.

Pruning Rule 6.5.3 *An object o cannot affect the safe zone if its range network RN_o does not intersect the safe zone.*

Proof Recall that the safe zone is given by $S = \cap_i RN_i - \cup_x RN_x$. If the range network of an external object x does not intersect the safe zone, it implies that the set difference of the current safe zone and the range network of x is the same as the current safe zone. Hence, the safe zone is not affected. ■

In Fig. 6.21, the object o_3 does not affect the safe zone because its range network does not intersect the safe zone. The next pruning rule is applicable to only the road networks where the weight of each edge corresponds to the length of the edge. For such networks, Euclidean distance between any two points is always smaller than or equal to the minimum road network distance between them.

Pruning Rule 6.5.4 *An object o cannot affect the safe zone S if $mindist(o, S) \geq r$ where $mindist(o, S)$ is minimum Euclidean distance of o from the safe zone S.*

Proof For any two points x and y, the Euclidean distance between them is always smaller than or equal to the minimum road network distance between them. Hence, if the minimum Euclidean distance between o and any point x of the safe zone is greater than r, it implies that its minimum road network distance from x is greater than r. In other words, the range network of o does not intersect the safe zone. ∎

Pruning Rule 6.5.5 *An external object o_j cannot affect the safe zone if $MinNetDist(o_i, o_j) \geq 2r$ where o_i is* any *internal object.*

This pruning rule is similar to the pruning rule 6.2.3. The proof of correctness is basically the same except that the term range network is to be used whenever the term circle appears in the proof of the pruning rule 6.2.3.

Pruning Rule 6.5.6 *Let d_{max} be the distance as defined in the description of pruning rule 6.5.2. An object o cannot affect the safe zone if $MinNetDist(q, o) \geq r + d_{max}$.*

Proof We prove this by showing that the range network of such an object o does not intersect the safe zone. Let x be a point in the safe zone. The minimum network distance between o and x satisfies $MinNetDist(o, x) \geq MinNetDist(q, o) - MinNetDist(q, x)$. We know that $MinNetDist(q, x) \leq d_{max}$ (by definition of d_{max}) and $MinNetDist(q, o) \geq r + d_{max}$. Hence, $MinNetDist(o, x) \geq (r + d_{max}) - d_{max} \geq r$. This implies that the range network of o does not contain any point x of the safe zone (i.e., its range network does not intersect the safe zone). ∎

In the example of Fig. 6.21, d_{max} is 6 and an object o cannot affect the safe zone if its minimum network distance from q is at least 16.

Before we present our final pruning rule, we define a few additional terms. We say that a vertex v is a *dead* vertex if its range network RN_v does not intersect the safe zone.

A path between two points a and b is called a *valid* path if the path does not contain *any* dead vertex. For example, the vertices b, c, e and f are dead vertices because, for each of these vertices, its minimum distance to the safe zone is larger than 10. The path $d \to z \to t$ is a valid path whereas the path $d \to b \to t$ is not a valid path.

Pruning Rule 6.5.7 *An object o cannot affect safe zone if there does not exist a valid path between o and q.*

Proof By definition, the safe zone is a connected network and the query q lies on it. Moreover, it follows from the definition of dead vertex that the safe zone cannot contain any dead vertex v. This implies that if there exists a valid path between o and any point of the safe zone x then there exists a valid path between o and q. Since we know that there does not exist any valid path between o and q, this means there does not exist any valid path between o and any point x of the safe zone. This implies that there always exists a dead vertex v on every path connecting o and x. Hence, $MinNetDist(o, x) > r$ because $MinNetDist(v, x) > r$ and the path from o to x passes through v. So o cannot affect the safe zone. ∎

For example, in Fig. 6.21, any object o that lies on the edge $e(b, c)$ cannot affect the safe zone because both the vertices b and c are dead and there does not exist a valid path between o and q.

We use this pruning rule in our algorithm while exploring the road network. A vertex that is marked dead is not further explored and hence the pruning rule limits the number of explored vertices.

6.5.3 Algorithm

Similar to Lemma 6.3.1, it can be shown that the order in which the objects are accessed is important. More specifically, an object o_i should be accessed before an object o_j if $|r - MinNetDist(q, o_i)| < |r - MinNetDist(q, o_j)|$ (the proof is similar to the proof of Lemma 6.3.1). For this reason, we use a min-heap H that gives priority to the objects with smaller $|r - MinNetDist(q, o_i)|$ (i.e., the key of each entry e of the heap is $|r - MinNetDist(q, e)|$.

Algorithm 14 **Network Range Query** (q, r)

Input: q: the query point; r: the range

Description:

```
1: initialize a min-heap $H$ /* key of each entry $n$ is to bet set to $|r - MinNetDist(q, n)|$ */
2: insert in $H$ vertices and objects lying on every edge that overlaps with $RN_q$
3: insert the objects lying on $RN_q$ in answer list
4: while $H$ is not empty do
5:   de-heap an entry $n$
6:   if $n$ is a vertex then
7:     if $RN_n$ does not intersect the safe zone then
8:       mark $n$ as dead;
9:     else
10:      for each adjacent vertex $v$ of $n$ do
11:        update/insert objects lying on edge $e(n, v)$ in $H$
12:        if $v$ is not marked dead then
13:          update/insert $v$ in $H$
14:   else if $n$ is an object and cannot be pruned then
15:     update the safe zone
16: return answer list and safe zone
```

Algorithm 14 presents the details of our technique. Initially, the objects and vertices that lie in the range network of q are inserted in the min-heap H. Then, the algorithm iteratively de-heaps the entries from the heap.

If the de-heaped entry n is a vertex and RN_n does not intersect the safe zone, we mark the vertex as dead. Otherwise, we process it as follows. For each of its adjacent vertices v, we insert in the heap the objects that are located on edge $e(n, v)$. It is possible that the objects on the edge $e(n, v)$ had already been inserted. In this case, for each object o lying on the edge $e(n, v)$ we update $MinNetDist(q, o)$ if its network distance from q via n is smaller than the previously stored $MinNetDist(q, o)$. Its key $|r - MinNetDist(q, o)|$ is also updated accordingly. Moreover, if the vertex v is not marked dead we insert v in the min-heap. If the vertex v already exists in the heap (e.g., it was inserted when another of its neighbors was considered), we update $MinNetDist(q, v)$ and its key $|r - MinNetDist(q, v)|$ accordingly.

Finally, if the de-heaped entry n is an object and it cannot be pruned by any of the pruning rules presented in the previous section, we update the safe zone using n and update d_{max}. The algorithm stops when the heap becomes empty.

6.5.4 Updating the safe zone

In this section, we present our technique for updating the safe zone (line 15 of Algorithm 14). We explain the main idea to update the safe zone for external objects. The technique for the internal objects is similar.

Recall that the safe zone is $S = \cap_i RN_i - \cup_x RN_x$ where the range network of every internal object i contributes to the intersection and the range network of every external object x contributes to the union. Hence, to update the safe zone for an external object, we need to delete the segments of the safe zone that lie within its range network.

Consider the example of Fig. 6.22(a) where the current safe zone consists of $e(q, b)$, $e(q, f)$ and $e(q, d)$ (shown in broken lines). Assume that an object o_1 is used for updating the safe zone ($r = 10$). The range network of o_1 is shown in Fig 6.22(b) (in thick lines). The segment $seg(d, k)$ lies within the range network of o_1 and can be removed from the safe zone. Fig. 6.22(b) shows the updated safe zone which consists of $e(q, b)$, $e(q, f)$ and $seg(q, k)$. Next we show that, to update the safe zone, we do not need to compute the complete range network for every object.

We define the *vertex flow* for a vertex v with respect to an object o as $F(v, o) = r - MinNetDist(v, o)$. In Fig. 6.22, assume that range r is 10. The vertex flow of c with

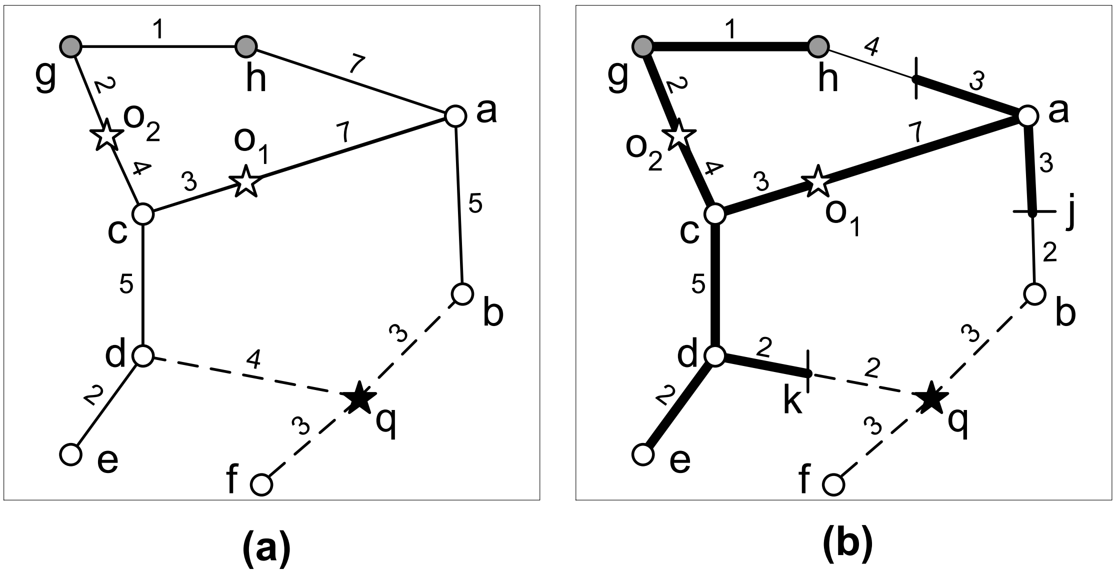


Figure 6.22: Updating the safe zone ($r = 10$)

respect to o_1 is $F(c, o_1) = 10 - 3 = 7$. Similarly, $F(c, o_2) = 10 - 4 = 6$. The *maximum vertex flow* $F_{max}(v)$ of a vertex v is the maximum of $F(v, o)$ over all objects o considered so far. In Fig. 6.22, the maximum vertex flow of c is 7 (i.e., $F_{max}(c) = 7$).

The vertex flow $F(v, o)$ denotes that every point p that lies within distance $F(v, o)$ of the vertex v lies on the range network RN_o of object o. For instance, when the range network of o_1 is computed in Fig. 6.22(b), it discovers everything within distance $F(c, o_2) = 7$ of the vertex c.

We now show that we do not need to compute the complete range network for an object o if its range network contains a vertex v such that $F(v, o) \leq F_{max}(v)$. Consider that the range network of object o_1 has been considered and $F_{max}(v) = F(c, o_1) = 7$. When the range network of the object o_2 is being computed, the vertex c is discovered and $F(c, o_2) = 6$. Since $F(c, o_2) < F_{max}(v)$, we do not need to further explore the range network by considering the adjacent vertices of c. This is because every point within distance 7 of c has already been discovered by the range network of o_1 (whereas the range network of o_2 will discover every point within distance 6 of c).

Now, we define another condition that avoids the complete computation of range network for certain objects. If a dead vertex v is discovered during the range network computation of an object o, we do not need to further explore the vertex v. By definition of a dead vertex v, its range network does not intersect the safe zone. This means that every point within distance r of v lies outside the safe zone. Hence, the range network of o that passes through the vertex v cannot affect the safe zone. In Fig. 6.22, the vertices g and h are the dead vertices. When the range network of o_2 is being computed, it does not need to further explore the vertex g. Recall that the vertex c was not required to be explored because $F(c, o_2) < F_{max}(c)$. Hence, during the range network computation of o_2, only the edge $e(g, c)$ is discovered.

6.6 Experiments

First we present the experimental results for our approach in Euclidean space. Then, in Section 6.6.6, we present the results for range queries in road networks. To evaluate the performance of our proposed approach, we compare our approach with an optimal algorithm and a naïve algorithm. We assume that the optimal algorithm already knows the safe zone and updates the results only when the query leaves the safe zone. To compute the initial results, the optimal algorithm visits the objects that lie within the range. To update the results, the algorithm searches only the area that may contain the new answers. We only consider the I/O cost for the optimal algorithm (the CPU time is assumed to be zero).

The naïve algorithm prunes every object o_i such that its circle does not intersect with the circle of any guard object. That is, an object or rectangle can be pruned if its distance from all guard objects is greater than $2r$.

All the experiments were conducted on Intel Xeon 2.4 GHz dual CPU with 4 GBytes memory. We used real data set as well as synthetic data set. The real data set[2] contains $175,813$ points of interests in North America that corresponds to a data universe of 5000Km×5000Km. To verify the theoretical analysis, we created synthetic data sets consisting $50,000$ to $150,000$ points following uniform distribution within the same data

[2]http://www.cs.fsu.edu/ lifeifei/SpatialDataset.htm

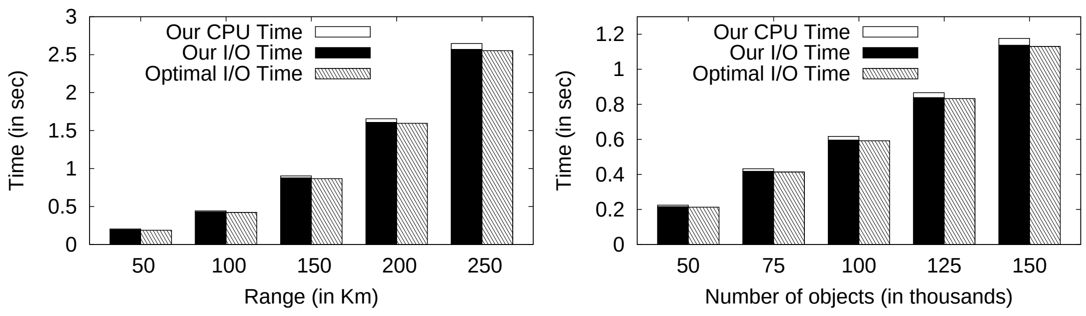
universe size. The objects are indexed by an R-tree with node size set to $2K$.

Parameter	**Range**
Number of objects ($\times$1000)	50, 75, **100**, 125, 150
Range (in Km)	50, 100, **150**, 200, 250
Average speed (in Km/hr)	40, 60, **80**, 100, 120

We simulated moving queries (moving cars) by using the spatio-temporal data generator described in [Bri02]. The average speed of moving queries varies from 40 Km/hr to 120 Km/hr. All queries are continuously monitored for 5 minutes and the results shown correspond to the average monitoring cost for a single query for the 5 minutes duration. All the experimental results shown correspond to the real data set except the results where we show the effect of number of objects. The table above shows the default parameters.

6.6.1 Cost comparison

The cost of each algorithm consists of I/O cost (by charging 2ms for each node access) and CPU cost (assumed zero for the optimal algorithm). The naïve algorithm was at least 20 times slower[3] than our algorithm for all settings so we exclude it from figures to better illustrate the comparison of our algorithm with the optimal algorithm.



(a) Radius (b) Number of objects

Figure 6.23: Efficiency

In Fig. 6.23 and Fig. 6.24, we compare the cost of our algorithm with the cost of the optimal algorithm for different ranges, different number of objects and varying speed.

[3]We also compared our algorithm with naïve algorithm for in-memory data and observed 30-70 times better performance. This shows that our proposed approach performs good even for in-memory computation models.

The performance of our algorithm is close to the optimal algorithm. The main cost for our proposed approach is the I/O cost which is very close to the I/O cost of the optimal solution. This shows that the overhead of computing the safe zone is very small compared to the cost of the range query.

6.6.2 Verification of the theoretical analysis

First, we study the escape probability and verify the theoretical results obtained. In our experiments, the escape probability of a query is computed by dividing the number of times it leaves the safe zone by the total number of movements recorded. We record the movement every second and check whether the query lies within the safe zone or not. Fig. 6.25 and Fig. 6.26 compare the escape probabilities with the theoretical results for different values of different parameters. Please note that Fig. 6.26 corresponds to the experiments run on the real data and it is evident that the theoretical results are accurate even on the real data.

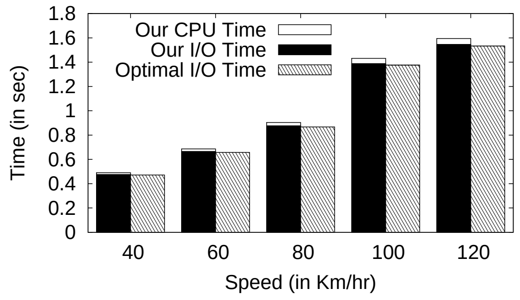


Figure 6.24: Efficiency (effect of speed)

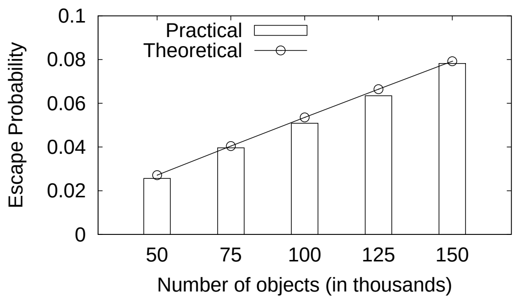
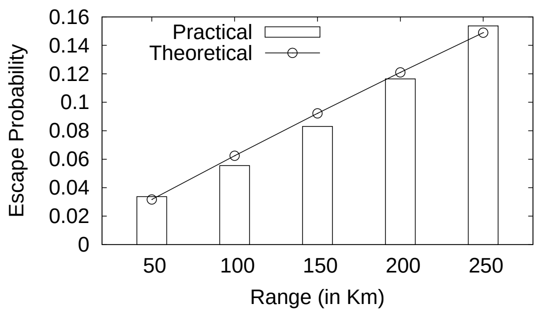


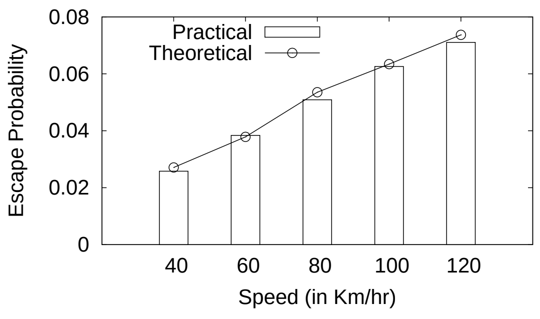
Figure 6.25: Escape probability vs data cardinality

As expected, the escape probability increases with the number of objects. The range and the speed have a similar effect on the escape probability. The results demonstrate that the escape probability is small, which shows the effectiveness of our proposed approach in real world settings.

In Fig. 6.27, we show the expected distance for queries run on the synthetic data set with increasing number of objects and increasing range of the query. It shows that the actual expected distance is close to the expected bounds we obtained in Section 6.4.



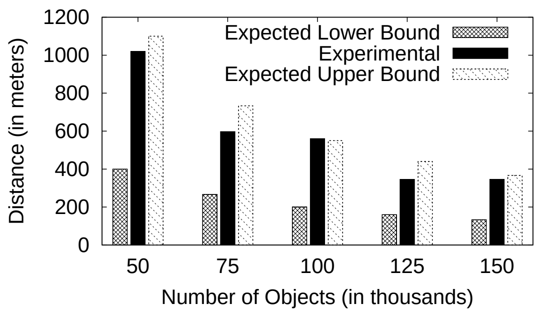
(a) Effect of range



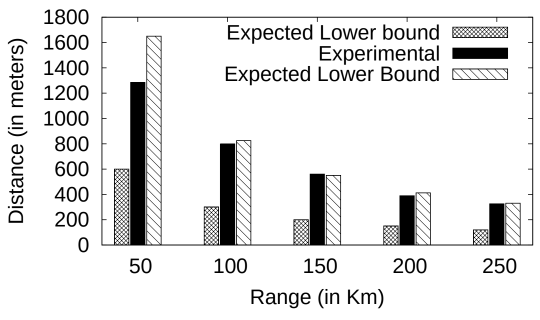
(b) Effect of speed

Figure 6.26: Escape probability

Moreover, the actual expected distance is from 300 meters to 1200 meters.



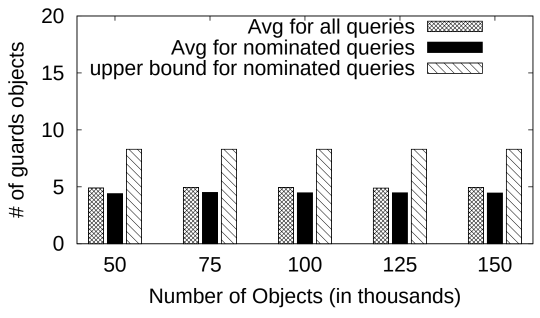
(a) Effect of data cardinality



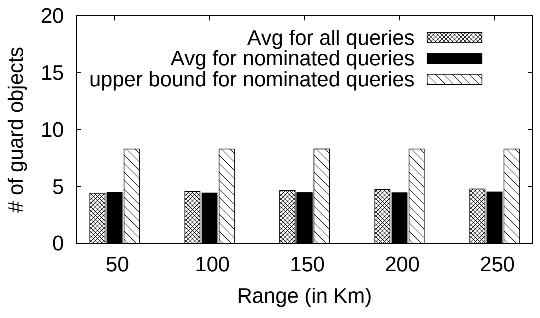
(b) Effect of range

Figure 6.27: Expected distance

Fig. 6.28 shows the average number of guard objects for all queries and compares the theoretical bound with the actual number of guard objects. As stated in Section 6.4, our theoretical upper bound is valid for the queries for which maximum distance to the safe zone is smaller than $C \cdot m_{up}$ where C is a constant. We observed that when C is set to 2, 30% to 50% queries satisfy the constraint. We call such queries the nominated queries.



(a) Effect of data cardinality



(b) Effect of range

Figure 6.28: Number of guard objects

In Fig. 6.28, we show the average number of guard objects for all queries as well as the average number of guard objects for the nominated queries. It is interesting to note that the average number of guard objects for all queries is around 5 regardless of the experiment settings.

6.6.3 Effectiveness of the proposed access order

In Fig. 6.29, we show the effectiveness of our proposed access order. We tried two other access orders namely *MinFirst* and *RandomAccess*. In MinFirst access order, the objects are accessed in increasing order of their distances from the query. In RandomAccess, the objects are accessed randomly. However, to improve the performance of RandomAccess, we give priority to the objects that lie within the range over the objects that lie too far from the query.

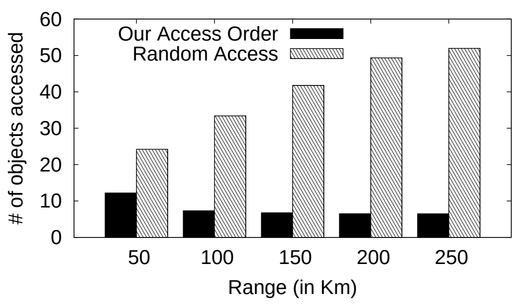


Figure 6.29: Effectiveness of access order

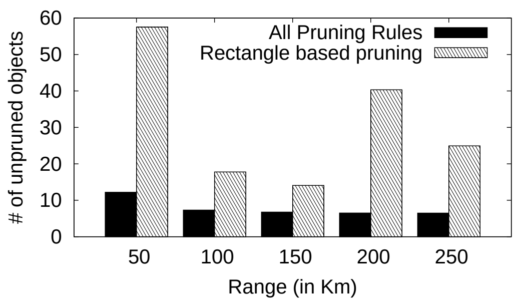


Figure 6.30: Effectiveness of pruning rules

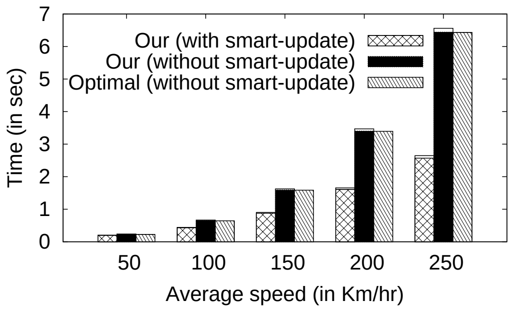
For each access order, we record the number of objects considered for updating the safe zone. MinFirst considers from 100 to 1300 objects when the range is increased from 50 Km to 250 Km. We exclude it from Fig. 6.29 to better illustrate the comparison of the other two access orders. Our proposed algorithm accesses around 6 objects when the range becomes larger. Note that an optimal access order will access only the guard objects (the number of guard objects is around 5). This shows that our proposed access order is close to the optimal access order.

6.6.4 Effectiveness of the pruning rules

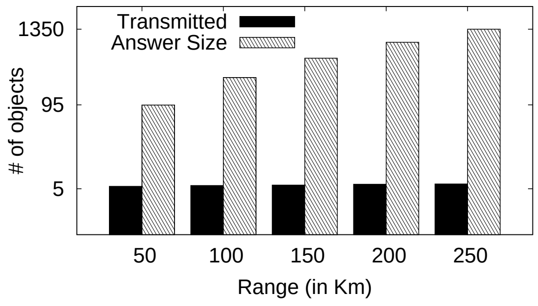
In Fig. 6.30, we show the effectiveness of the rectangle based pruning rules and the guard objects based pruning rules. As expected, although the rectangle based pruning rule is computationally cheap, it is unable to prune many objects. On the other hand, the guard objects based pruning rules are more effective.

6.6.5 Effectiveness of Smart-Update

Fig. 6.31 shows the effectiveness of our proposed smart-update. In Fig. 6.31(a), we show the cost of our algorithm with and without using the smart-update. We also show the performance of the optimal algorithm if the smart-update is not applied, i.e., every time a query leaves the safe zone, the optimal approach without the smart-update accesses all the objects within the range and sends to the client. The effectiveness of our proposed smart-update is evident from Fig. 6.31(a). As the range increases, the performance gain by the smart-update increases because it avoids to visit a larger area.



(a) Effect on Cost



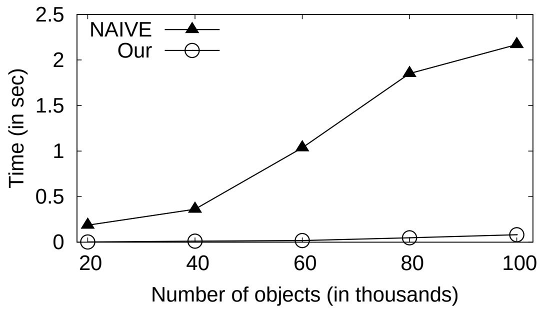
(b) Effect on data transmission

Figure 6.31: Effectiveness of the smart-update

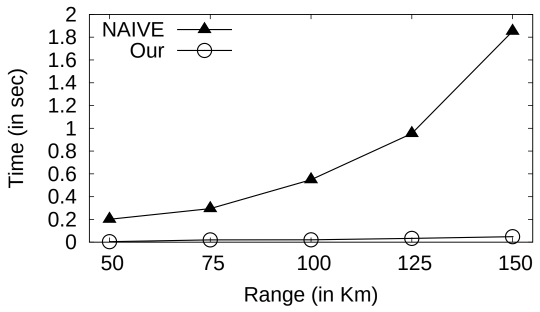
Fig. 6.31(b) shows the average number of objects transmitted to the query whenever the server receives an update request. It also shows the total number of objects that lie within the range (shown as answer size). Please note that a log scale is used to better illustrate the trend. If the results are updated without using the smart-update, all the objects that lie within the range are to be sent again. Using our proposed smart-update approach, the number of objects that are sent to client are around 5. Note that this number includes the number of guard objects that are sent to the client.

6.6.6 Range queries in road networks

We use the road map of California[4] that consists of 21694 road segments (edges). We generated queries moving with default speed of 80 Km/hr. Each query starts at a randomly chosen vertex. Whenever the query reaches at a vertex, one of its adjacent vertex is randomly chosen as destination and the query continues travelling. Each query is monitored for 5 minutes and the reported time is the total time for the 5 minutes duration of a query. Naïve algorithm recomputes the results whenever the query reports location update. In Fig. 6.32, we change the number of objects and the range of the query and observe that our approach is up to two orders of magnitude faster than the Naïve algorithm.



(a) Effect of data size

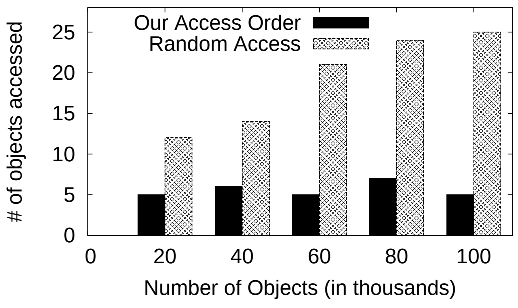


(b) Effect of range

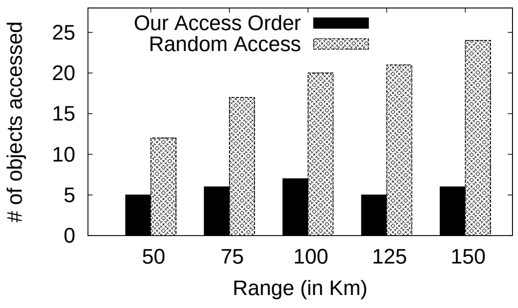
Figure 6.32: Range queries in a road network

In Fig. 6.33, we show the effectiveness of our proposed access order. Similar to the experiments for Euclidean distance based queries, we observe that our access order performs better than *MinFirst* access order and the random access order. MinFirst access order was outperformed by both the random access order and our access order so we do not include it in Fig. 6.33.

[4]http://www.cs.fsu.edu/ lifeifei/SpatialDataset.htm



(a) Effect of data size



(b) Effect of range

Figure 6.33: Effectiveness of access order

6.7 Summary

In this chapter, we focus on the distance based range queries that continuously change their locations in a Euclidean space. We present an efficient and effective monitoring technique based on the concept of a safe zone. The safe zone of a query is the area with a property that while the query remains inside it, the results of the query remain unchanged. Hence, the query does not need to be re-evaluated unless it leaves the safe zone. Our contributions are as follows. 1) We propose a technique based on powerful pruning rules and a unique access order which efficiently computes the safe zone and minimizes the I/O cost. 2) We theoretically determine and experimentally verify the expected distance a query moves before leaving the safe zone and, for majority of queries, the expected number of guard objects. 3) Our experiments demonstrate that the proposed approach is close to optimal and is an order of magnitude faster than a naïve algorithm. 4) We also extend our technique to monitor the queries in a road network. Our algorithm is up to two order of magnitude faster than a naïve algorithm.

Chapter 7

A Unified Algorithm to Answer Top-k Pairs Queries

In this chapter, we study a generalized version of k closest pairs query problem called top-k pairs query. We provide a unified framework to answer a broad class of top-k pairs queries including k closest and k-furthest pairs queries and their variants. This chapter is based on our research reported in [CLW+11].

7.1 Overview

Given a set of objects $\{o_1, \cdots, o_N\}$ and a ranking function that returns the score of a pair of objects (o_u, o_v), a top-k pairs query returns k pairs with the best scores. An important and well studied special case of the top-k pairs query is the k closest pairs query which returns k pairs with the smallest distances. The k closest pairs queries have been extensively studied in the context of computational geometry (see [Smi97] and references therein).

The database community has also conducted significant research on the k closest (or most similar) pairs queries, k furthest (or most dissimilar) pairs queries and their variants [HS98, CMTV00, YL02, SZS03]. However, all the existing techniques are developed to solve some specific problems and there does not exist a unified approach that answers different variants of the top-k pairs queries (e.g., different L_p distances). Another interesting variation for which no efficient solution exists is to find the pairs of the objects that are similar to each other in one subspace and dissimilar in another subspace. We are the first to provide a unified framework that supports a broad class of top-k pairs queries including the above mentioned queries.

We present a unified approach to efficiently answer the top-k pairs queries based on generic scoring functions which are not supported by the existing work. Consider a simple example of an insurance company. The manager might want to retrieve two insurance agents who sell very similar amount of policies (i.e., the total premium of their sold policies is similar) but receive very different salaries. Suppose that the relevant information is stored in a table named `agent`. The manager may issue the following query to retrieve the top-k pairs of agents.

```
Q1: select a.id, b.id from agent a, agent b
where a.id < b.id
order by
|a.sold - b.sold| - |a.salary - b.salary|
limit k
```

Here $|x - y|$ denotes the absolute difference of x and y. Note that the `order by` clause prefers the pair of agents with larger difference in their salaries and smaller difference in the amount of the policies they sold[1]. The condition $a.id < b.id$ is used to avoid the pair (a, b) being repeated as (b, a).

While the example shows a simple ranking criteria, in the real applications, the users may define more sophisticated scoring functions. Our framework allows the users to define a different scoring function for each attribute involved in the query. Such scoring functions are called *local* scoring functions. The users define a *global* scoring function

[1]Without loss of generality, we assume that the top-k pairs queries retrieve k pairs with the smallest overall scores.

that computes the final score of a pair by combining its scores on all attributes.

Our framework supports any global scoring function that is *monotonic* and any local scoring function that is *loose monotonic*. A wide range of functions that are used in many real applications are monotonic. Although we define monotonic and loose monotonic scoring functions in Section 7.2.1, we remark here that the loose monotonic functions are more general than the monotonic functions. In the above example, the two local scoring functions are $|a.sold - b.sold|$ and $-|a.salary - b.salary|$, respectively. The global scoring function is the sum of the local scores.

Our framework does not fix the number of attributes involved in the query. In other words, the users can issue a top-k pairs query on any subset of the attributes using a different loose monotonic scoring function for each attribute. This enables us to support many interesting queries (e.g., similarity in one subspace and dissimilarity in another).

We further generalize the supported top-k pairs queries by classifying them into *chromatic* and *non-chromatic* top-k pairs queries. The *chromatic* queries are further classified into *homochromatic* and *heterochromatic* top-k pairs queries. Suppose that each object in the database has been assigned a color. A homochromatic top-k pairs query returns the top-k pairs among the pairs that contain two objects having the same color. On the other hand, a heterochromatic top-k pairs query considers only the pairs that contain two objects having different colors. A top-k pairs query that does not consider the colors of the objects (i.e., all pairs are considered) is called a non-chromatic top-k pairs query.

In the query Q1, the user may want to consider only the pairs of agents who work under different managers. The user may issue a heterochromatic top-k pairs query by adding the condition `a.manager` $\neq$ `b.manager` in the `where` clause of the query. Note that the heterochromatic queries are more general than the *bichromatic* queries. The bichromatic queries assume that some of the objects are assigned blue color and others are assigned red color. Only the pairs that contain one red object and one blue object are considered. Existing work on k closest pairs queries [HS98, CMTV00] solve bichromatic queries and the extension to heterochromatic queries is either non-trivial or inefficient.

Below, we summarize our contributions.

- We are first to provide a unified and efficient approach for a broad class of top-k pairs queries. Our framework does not require any pre-built data structure, has low memory requirement and is easy to implement.
- We theoretically analyse the performance of the proposed algorithms and show that the expected performance is optimal when the number of attributes involved is two or less[2].
- Our extensive experiments demonstrate a significant improvement over the existing best known solution for k closest pairs query. For the more general top-k pairs queries, we compare our algorithm with a naïve algorithm and observe up to three orders of magnitude improvement.
- Due to the generality of the framework, it can support several other interesting queries (e.g., *skyline pairs*, *rank-based* top-k pairs, and *exclusive* top-k pairs queries). In Section 7.6, we present efficient solutions for both the chromatic and non-chromatic variants of these queries and provide a detailed theoretical analysis.

The rest of the chapter is organized as follows. In Section 7.2, we formally define the problem. We present our framework and its advantages in Section 7.3. In Section 7.4, we present our technique to create and maintain internal memory and external memory sources which is the core part of our approach. We present our query processing algorithm in Section 7.5. In Section 7.6, we present techniques to answer other variants of top-k pairs queries. Experiment results are given in Section 7.7. Section 7.8 summarizes this chapter.

[2]When d attributes are involved, the expected time complexity is $O(dV^{\frac{d-1}{d}}k^{\frac{1}{d}}Log\ N)$ and expected IO cost is $O(\frac{d}{B}V^{\frac{d-1}{d}}k^{\frac{1}{d}}(Log_{\frac{M}{B}}\frac{N}{B}))$ where V is the total number of valid pairs, N is the total number of objects, B is the number of pairs that can be stored in one disk block and M is the number of pairs that can be stored in the main memory.

7.2 Preliminaries

7.2.1 Problem Definition

First, we define *monotonic* and *loose monotonic* scoring functions. A function f is called a monotonic function if it satisfies $f(x_1, \cdots, x_n) \leq f(y_1, \cdots, y_n)$ whenever $x_i \leq y_i$ for every $1 \leq i \leq n$.

Now, we define the loose monotonic functions. Let $s(.,.)$ be a scoring function that takes two values as parameter and returns a score. A function $s(.,.)$ is a loose monotonic function if for every value x_i both of the following are true: i) for a fixed x_i and every $x_j > x_i$, $s(x_i, x_j)$ either monotonically increases or monotonically decreases as x_j increases, and ii) for a fixed x_i and every $x_k < x_i$, $s(x_i, x_k)$ either monotonically increases or monotonically decreases as x_k decreases.

The absolute difference of two values (e.g., $|x_i - x_j|$) is a loose monotonic function. This is because for a fixed x_i and any value x_j larger than it, the absolute difference monotonically increases when x_j increases. Similarly, for any fixed x_i and any value x_k smaller than it, the absolute difference monotonically increases as x_k decreases. Please note that the loose monotonic functions are more general because these require the scores to be monotonic only with respect to every individual x_i and the function may not be monotonic in general. All monotonic functions are loose monotonic functions but the converse may not be true for some functions. For example, the absolute difference of two values is a loose monotonic function but it is not a monotonic function. The average of two values is a loose monotonic function as well as a monotonic function.

For ease of presentation, we classify loose monotonic functions into different categories. A loose monotonic function is called right increasing (resp. decreasing) function if for every $x_j > x_i$ for the fixed x_i, $s(x_i, x_j)$ monotonically increases (resp. decreases) as x_j increases. For example, the absolute difference is a right increasing function. A loose monotonic function is called left increasing (resp. decreasing) function if for every $x_k < x_i$ for the fixed x_i, $s(x_i, x_k)$ monotonically increases (resp. decreases) as k decreases.

For instance, the absolute difference is a left increasing function whereas the average of two values is a left decreasing function.

Let d be the number of attributes specified by the user for a top-k pairs query. For each attribute i, the user specifies a loose monotonic scoring function $s_i(.,.)$ that computes the score of a pair on the attribute i. Such scoring function is called a local scoring function and the score $s_i(a,b)$ of a pair (a,b) is called its local score. The users are allowed to define a different local scoring function for each attribute. The user defines a monotonic global scoring function f that takes d local scores as parameter and returns the final score $SCORE(a,b)$ of a pair (a,b) as $f(s_1(a,b), \cdots, s_d(a,b))$.

Score-based top-k pairs query. Given a set of objects O, a non-chromatic top-k pair query returns a set of pairs $P \subseteq O \times O$ that contains k pairs such that for any pair $(a,b) \in P$ and any pair $(a',b') \notin P$, $SCORE(a,b) \leq SCORE(a',b')$.

Chromatic queries. Consider that each object in a set of objects O is assigned a color. A chromatic query is similar to a non-chromatic query except for an additional constraint; that is, only the pairs that meet the color requirement are considered. A homochromatic top-k pairs query considers only the pairs that have two objects having the same color. In contrast, a heterochromatic top-k pair query considers only the pairs that contain objects with different colors.

We define the skyline pairs query, the rank-based top-k pairs query, and the exclusive top-k pairs queries in Section 7.6.1, Section 7.6.2, and Section 7.6.3, respectively.

7.2.2 Top-k Query Processing

Our algorithm uses some existing top-k query processing algorithms that combine multiple ranked sources and return the top-k objects. Therefore, in this section, we briefly describe these algorithms. These algorithms assume that each source S_i contains the objects ranked on their scores according to a preference i. Let x_i be the score of an object in a source S_i. The final score of the object is computed by using a monotonic function $f(x_1, \cdots, x_d)$ where d is the number of sources. The algorithms report k objects with the smallest final scores.

The top-k algorithms assume that the objects in a source can be accessed in two ways. A *sorted* access on a source reads the next object in the sorted order. A *random* access returns the score of any specified object in a given source. In a random access, the specified object is searched in the source and its score is returned. It is important to note that not all the sources can support both types of accesses (e.g., a search engine provides the sorted access but does not support a random access).

Now, we briefly introduce three well known algorithms.

Fagin's Algorithm (FA). FA [FLN03] assumes that the sources support both sorted and random accesses. Let there be d sources $S_i, \cdots, S_d$. FA works as follows.

1. Do sorted access in parallel on each of the d sources. Go to step 2 when there are at least k objects that have been returned by *every* source.

2. For each object that has been returned by at least one source, do the random accesses on the other sources to retrieve its scores on remaining sources and compute its final score. Return k objects with the smallest final scores.

A major problem with FA is that it uses unbounded buffer (i.e., the number of objects stored in the main memory may be arbitrarily large).

Threshold Algorithm (TA). TA (independently proposed in [FLN03, NR99, GBK00]) also assumes that the sources support both sorted and random accesses. TA works as follows.

1. Do sorted accesses in parallel on each of the d sources. For each object o returned from a source S_i, do the random accesses on every other source to obtain its scores in the other sources. Compute the final score of o using the monotonic function f. Maintain a heap that contains k objects with the smallest scores. Let W_k be the largest of the scores of the objects maintained in the heap.

2. Let $\underline{x}_i$ be the score of the last object returned from the source S_i through a sorted access. After every sorted access, update the *threshold value* as $t = f(\underline{x}_1, \cdots, \underline{x}_d)$. Terminate the algorithm when $t \geq W_k$. Report the objects in the heap as top-k objects.

It has been shown that the number of accesses by TA cannot be larger than the number of accesses by FA. Furthermore, TA is optimal in number of accesses when every source supports both the sorted and random accesses. Moreover, the buffer size of TA is $O(k)$ because at any time it keeps only the best k objects in its buffer.

No Random Access (NRA) Algorithm. NRA [FLN03] assumes that the sources do not support the random accesses. The algorithm works as follows.

1. Do the sorted accesses in parallel on each of the d sources. For each returned object o, compute its best possible score $B(o)$ and its worst possible score $W(o)$ by assuming the best and worst possible scores on the sources that have not yet returned it. Maintain a heap that contains k objects with the smallest worst possible scores $W(o)$.

2. Let W_k be the largest of the worst possible scores of k objects in the heap. At each sorted access, update W_k and the best possible score $B(o)$ of every seen object o. Terminate the algorithm when $B(o) \geq W_k$ for every seen object o. Report the objects in the heap as the top-k objects.

It has been shown that NRA is optimal in the number of accesses when the random access is not supported by the sources. However, like FA, it also requires an unbounded buffer. Moreover, the best possible scores of all seen objects are to be updated whenever an object is returned by a sorted access.

Mamoulis et al. [MYCC07] present several interesting observations and propose an algorithm LARA that significantly improves the performance of NRA. LARA consists of two phases. In *growing phase*, the objects that are returned by the sorted accesses form a candidate set. They prove that no candidate object can be pruned during the growing phase. Hence, update of the best possible scores is not required. Let $\underline{x}_i$ be the last score seen on a source S_i and W_k be the k^{th} smallest worst score of the seen objects. The growing phase completes when $t \geq W_k$ where $t = f(\underline{x}_1, \cdots, \underline{x}_d)$ and denotes the best possible score of any unseen object.

In *shrinking phase*, the candidates are divided in 2^d categories based on the sources on which they have been seen. For each group, the candidate with the smallest worst score is called the leader. They prove that only the best possible scores of the leaders are to be updated whenever a new object is returned by a sorted access. If the leader of a group can be pruned, the whole group is pruned. The algorithm stops when there is no leader with its best possible score smaller than W_k.

7.3 Our Proposed Framework

Let d be the number of local scoring functions involved in the top-k pairs query. We map our problem to the well studied problem of top-k query that combines the scores from different *ranked sources* (see the previous section). More specifically, we maintain d sources (please see Fig. 7.1) such that each source S_i incrementally returns the pair with the best score according to the i^{th} local scoring function. The existing top-k algorithms (e.g., FA, TA and NRA) view these sources as the ranked inputs and can be used to retrieve the top-k pairs by combining these ranked inputs.

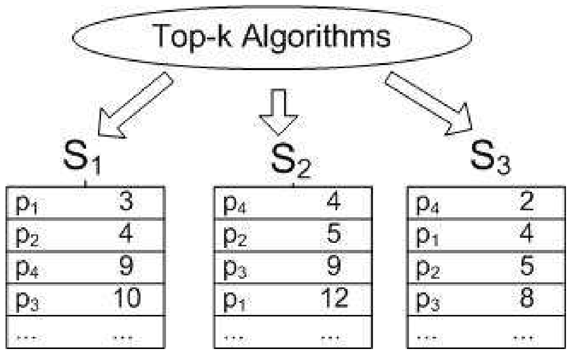


Figure 7.1: Our framework

Most of the existing work on the top-k queries can be applied to solve the problem of the top-k pairs queries. However, these algorithms assume that the sources can report the elements in a sorted order. Hence, it is important to develop efficient techniques to create and maintain the sources such that each source can return the pairs of objects in a sorted order. A straight forward solution to create a source S_i is to sort all the possible pairs according to their local scores on the i^{th} attribute. However, this solution requires storing and sorting $O(V)$ pairs where $O(V)$ is the number of valid pairs (this number is

$O(N^2)$ for non-chromatic queries if N is the number of objects). Clearly, the time and the space complexity of this straight forward approach may be prohibitive.

In the next section, we present an optimal internal memory algorithm and an optimal external memory algorithm to create and maintain such sources. The internal memory algorithm uses $O(N)$ space and is optimal in time complexity. The external memory algorithm is I/O optimal.

Below we highlight a few advantages of our framework.

1. No pre-built indexes required. Our proposed algorithm does not require any pre-built indexes, i.e., there does not exist any index at the time a query is issued. We remark here that the indexes like R-tree usually index all the dimensions (i.e., attributes) of the objects and the queries that involve a subset of these dimensions may not be answered efficiently by these indexes. Moreover, the pruning rules used on these indexes are based on the distance metrics and may not work for the generic scoring functions.

2. Known memory requirement. The existing techniques for k-closest pairs queries [HS98, CMTV00] use heap to store the intermediate nodes of the R-trees. The size of the heap may become as large as $O(V)$ and the system may run out of memory. In contrast, our external memory algorithm has a bounded memory requirement (it requires $O(k)$ space in addition to $2d$ buffer pages).

3. Efficient. Although our proposed approach supports more general top-k pairs queries and does not require any pre-built indexes, our experimental results demonstrate that the proposed approach is in general more efficient than the existing solutions of k closest pairs queries. We also conduct theoretical analysis and show that the expected cost of our proposed approach is optimal for the queries that involve two or less attributes.

4. Easily extendible. Due to the generality of the framework, it can be used to answer other types of queries. For example, a variation of FA [FLN03] algorithm on top of our ranked sources can be used to solve *skyline pairs* query that returns every pair that is not worse than any other pair. In Section 7.6, we demonstrate the extensibility of our framework by showing that it can efficiently solve several interesting queries like *skyline pairs*, *rank-based* top-k pairs, and *exclusive* top-k pairs queries.

5. Feasible for implementation in a DBMS. Unlike the existing techniques that target specific problems, our general algorithmic framework solves a broad class of top-k pairs queries (including all the existing variants) and is easy to implement. Moreover, the proposed technique outperforms existing algorithms both theoretically and experimentally. Hence, it is a good choice to be implemented in any DBMS.

7.4 Maintaining The Sources

7.4.1 Internal Memory Source

First, we define some terminologies. Suppose that all the objects are sorted in ascending order of their attribute values such that $o_1 \le o_2 \le \cdots \le o_N$. For any pair (o_u, o_v), we refer to the first object o_u in the pair as *host* and the second object o_v as *guest*. A pair (o_u, o_v) means that the object o_u is a host to a guest o_v.

For the ease of presentation, we assume that the local scoring function $s(.,.)$ satisfies[3] $s(o_u, o_v) = s(o_v, o_u)$. To avoid reporting a pair (o_u, o_v) again as (o_v, o_u), we will consider only the pairs (o_u, o_v) such that $u < v$. This implies that every object o_u can host only the objects that are on the right side of o_u in the sorted list $o_1 \le o_2 \le \cdots \le o_N$. For chromatic queries, only the objects that meet the color requirement and are on the right side of o_u will be considered its guests. Let o_v and $o_{v'}$ be two guests of o_u. We say that o_v is a better guest of o_u than $o_{v'}$ if $s(o_u, o_v) < s(o_u, o_{v'})$. An object o_v is called the best guest of a host o_u if for *every* other guest $o_{v'}$ of the host o_u, $s(o_u, o_v) \le s(o_u, o_{v'})$. We say that an object o_u has hosted the object o_v, if the pair (o_u, o_v) has been reported to the main algorithm.

Algorithm 15 presents the details of creating and maintaining a source. Initially, all the objects are sorted in ascending order of their attribute values (ties are broken

[3]The scoring functions for which $s(o_u, o_v) \ne s(o_v, o_u)$ can be easily handled by joining two sources such that the first source considers only the pairs (o_u, o_v) for every $u < v$ and the second source considers only the pairs (o_v, o_u) for every $u < v$.

Algorithm 15 Creating and maintaining a source

```
InitializeSource()
1: sort the objects in ascending order of their values
2: for each object $o_u$ do
3:     $o_v \leftarrow$ the best guest of $o_u$
4:     insert the pair $(o_u, o_v)$ into heap with score $s(o_u, o_v)$

getNextBestPair()
1: get the top pair $(o_u, o_v)$ from the heap
2: if next best guest of $o_u$ exists then
3:     $o_{v'} \leftarrow$ the next best guest of $o_u$
4:     insert the pair $(o_u, o_{v'})$ in heap with score $s(o_u, o_{v'})$
5: return $(o_u, o_v)$
```

arbitrarily). Then, for each object o_u, a pair (o_u, o_v) is created such that o_v is the best guest of o_u. All these pairs are inserted in a heap.

Whenever a request for the next best pair arrives, the source retrieves the top pair (o_u, o_v) from the heap and reports it to the main algorithm. The next best pair $(o_u, o_{v'})$ is inserted in the heap where $o_{v'}$ is the next best guest of o_u. At any stage during the execution, the next best guest of o_u is the best guest among the guests of o_u which has not been hosted by o_u earlier.

Example 7.4.1 *Consider the example of Fig. 7.2 which shows six objects o_1 to o_6 sorted on their attribute values. The values inside the circles are the attribute values. Assume that the scoring function is the absolute difference. A pair (o_u, o_v) is shown by a directed edge from the host o_u to the guest o_v. Initially, for each object, a pair with its best guest is created and inserted in the heap. Note that the best guest of an object is its right adjacent object when the function is absolute difference. Fig. 7.2(a) shows the pairs (see the edges) that are inserted in the heap. The number on an edge corresponds to the score of the pair. The best pair is (o_3, o_4) and its score is 1. When this is retrieved, the*

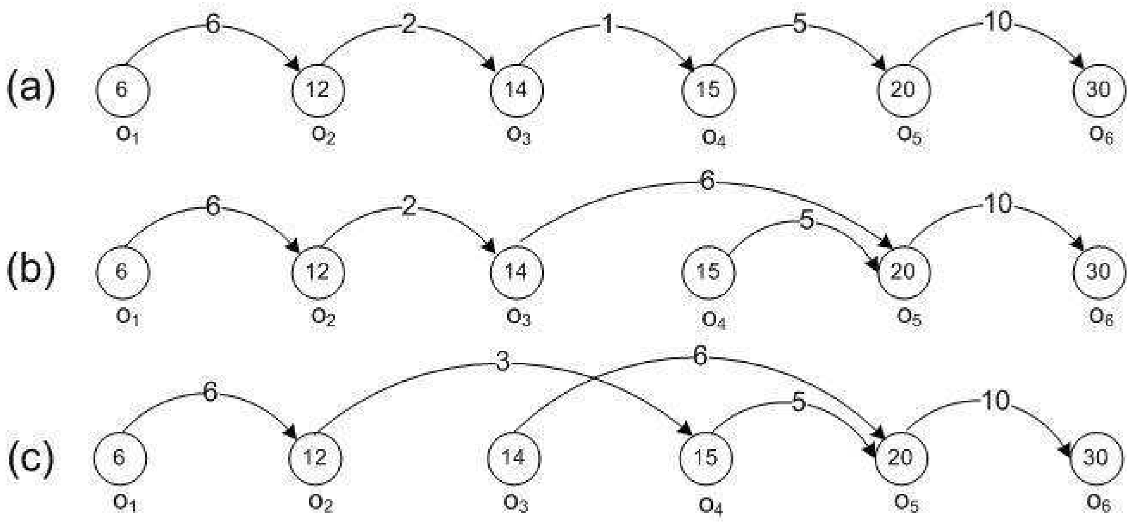


Figure 7.2: Illustration of Algorithm 15

algorithm determines that the next best guest of o_3 *is* o_5 *and inserts* (o_3, o_5) *in the heap with score 6 (see Fig. 7.2(b)). Now the top pair of the heap is* (o_2, o_3) *which is returned when the system requests the next best pair from this source. The next best guest of* o_2 *is* o_4 *so a new pair* (o_2, o_4) *is inserted in the heap with score* 3 *(see Fig. 7.2(c)).*

The intuitive justification of the correctness of the algorithm is that at any stage, we keep the best guests (among those that it has not hosted yet) for each object in the heap. This implies that for every pair that does not exist in the heap either there exists a better pair in the heap or the pair has already been reported to the main algorithm. The following lemma proves the correctness of the algorithm.

Lemma 7.4.2 *For any pair* (o_x, o_y) *that is not present in the heap and has not been reported earlier, there exists at least one pair* (o_u, o_v) *in the heap such that* $s(o_u, o_v) \leq s(o_x, o_y)$.

Proof First we prove it for the case when $x < y$. For each object o_x, we always have one object o_v in the heap (if o_x has not already hosted all valid guests) such that o_v is its best guest among the objects that it has not hosted yet. If o_x has hosted all valid guests, this implies that the pair (o_x, o_y) has been hosted. Otherwise, there must be at least one pair (o_x, o_v) in the heap such that $s(o_x, o_v) \leq s(o_x, o_y)$. This is because an object o_x will not host o_y unless it has hosted all the guests that are better than o_y.

Now, assume $x > y$. Following the similar argument as above, if the pair (o_y, o_x) has not been reported then there exists at least one pair (o_y, o_v) in the heap such that $s(o_y, o_v) \leq s(o_y, o_x)$. ∎

In order to achieve the optimal complexity, the algorithm must find the best guests for all N objects in O(N). Moreover, the algorithm must find the next best guest of any object o_u in O(1).

Before we show the details of how to do these operations with required complexity, we introduce the concept of *left adjacent* and *right adjacent* objects. A left (resp. right) adjacent object of o_u is the first object o_x on the left (resp. right) side of o_u in the sorted list $o_1 \leq o_2 \leq \cdots \leq o_N$ such that the pair (o_u, o_x) satisfies the color requirement.

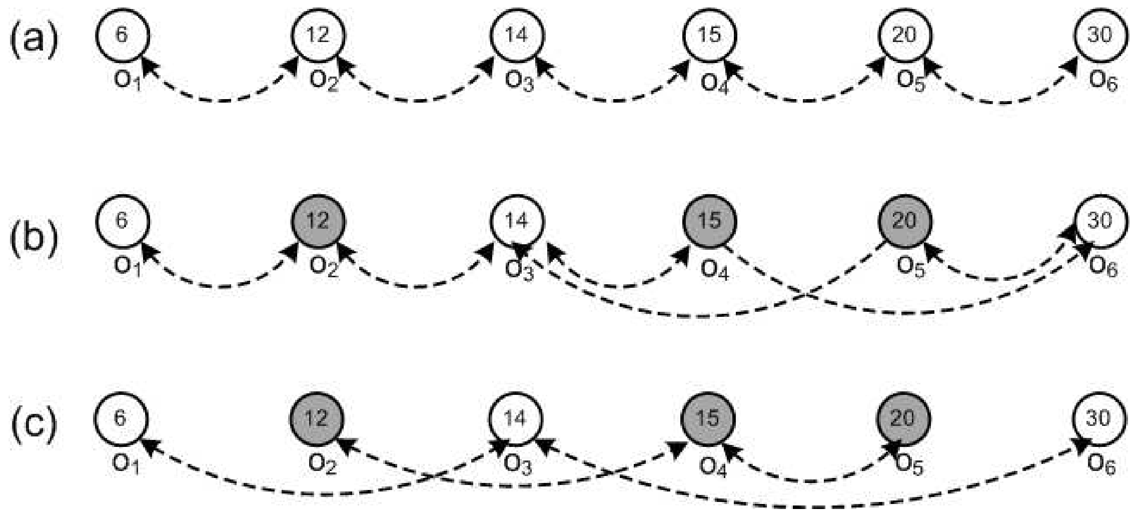


Figure 7.3: (a) Non-chromatic (b) Heterochromatic (c) Homochromatic

Fig. 7.3 shows an example where the objects o_1 to o_6 are shown. Some objects are shaded (o_2, o_4 and o_5) and others are white (o_1, o_3 and o_6). Fig. 7.3(a), (b) and (c) show the adjacent objects for non-chromatic queries, heterochromatic queries and homochromatic queries, respectively. The adjacent objects are shown with broken lines. An arrow from an object o_x to o_y indicates that o_y is the adjacent object of o_x in that direction.

Later in this section, we show that the left and the right adjacent objects of all the objects can be determined in O(N).

Finding the best guest for each object o_u

Below, we describe the procedure for the right increasing and the right decreasing functions (see Section 7.2.1 for the definitions).

For right increasing functions. Recall that if the scoring function is right increasing then the score $s(o_u, o_v) \leq s(o_u, o_{v'})$ if $v < v'$ (i.e., $o_{v'}$ is on the right side of o_v in the sorted list). Hence, for any object o_u, its best guest is its right adjacent object. For example, in Fig. 7.3(c), o_3 is the best guest of o_1 if the scoring function is right increasing function (e.g., absolute difference).

For right decreasing functions. For any object o_u, the best guest in this case is the right most object o_v such that the pair (o_u, o_v) meets the color requirement. More specifically, for non-chromatic queries, the best guest of any object o_u is $o_{_N}$. For example, in Fig. 7.3(a) the best guest of every object is o_6 if the scoring function is a right decreasing function (e.g., $s(o_u, o_v) = -(o_u + o_v)$).

For the heterochromatic queries, if $o_{_N}$ has a color different than o_u then $o_{_N}$ is the best guest of o_u. Otherwise the left adjacent object of $o_{_N}$ is the best guest of o_u because it is guaranteed to have a color different than o_u. In the example of Fig. 7.3(b), o_6 is the best guest of o_2, o_4 and o_5 whereas o_5 is the best guest of o_1 and o_3.

For the homochromatic queries, we scan the sorted list $o_1 \leq \cdots \leq o_N$ once and maintain the right most object of each color. For each object o_u, its best guest is the right most object of the same color. In the example of Fig. 7.3(c), o_6 is the best guest for o_1 and o_3 whereas o_5 is the best guest of o_2 and o_4.

Finding next best guest of any object o_u

Let o_v be the current best guest of the object o_u. The next best guest of o_u can be determined in O(1). Below, we describe how to find the next best guests for the right increasing functions and the procedure is similar for the right decreasing functions.

For the non-chromatic queries and the homochromatic queries, the next best guest $o_{v'}$ for an object o_u is the right adjacent object of o_v. In the example of Fig. 7.3(c), let o_3 be the current guest of o_1. The next best guest of o_1 is o_6 which is the right adjacent object of o_3.

For the heterochromatic queries, the next best guest of o_u is o_{v+1} if o_{v+1} has a color different than o_u. Otherwise, the right adjacent object of o_{v+1} is guaranteed to have a different color and hence is the next best guest of o_u. Consider the example of Fig. 7.3(b) and assume that the current best guest of the object o_2 is o_3. When (o_2, o_3) is reported, the algorithm checks o_4 to see if it is the next best guest of o_2. Since o_2 and o_4 have the same color, the next best guest of o_2 is o_6 which is the right adjacent object of o_4.

Finding the adjacent objects

Now we illustrate how to add pointers to the adjacent objects in O(N) time. For the non-chromatic queries, the procedure is trivial. So, we first discuss the procedure for determining the right adjacent objects for the heterochromatic queries. The procedure starts with setting the right adjacent object of o_N to NULL. Then, it starts scanning the sorted list of the objects from right to left. For each object o_u, if o_{u+1} has a different color than o_u then o_{u+1} is set as the right adjacent object of o_u. Otherwise, the right adjacent object of o_{u+1} is set as the right adjacent object of o_u.

Consider the example of Fig. 7.3(b). The right adjacent object of o_6 is set to NULL. The right adjacent object of o_5 is o_6 because they have different colors. The right adjacent object of o_4 is not o_5 because they have same color. So, the right adjacent object of o_5 (which is o_6) is set as the right adjacent object of o_4. The algorithm continues in this way. The left adjacent objects can be set similarly by scanning the list from left to right.

For the homochromatic queries, we assign the right adjacent objects as follows. While we scan the list from right to left, we maintain the last seen object of each color. For any object o_u, its right adjacent object is the last seen object of the same color (NULL if no object has been seen of this color). The left adjacent objects are set similarly by scanning the list from left to right.

Complexity

The first pair is returned in $O(N\ Log\ N)$ (the objects are sorted and $O(N)$ pairs are inserted in the heap). We remark that this meets the lower bound of returning the closest pair in one dimension [BO83]. Since our general framework covers the closest pairs, the lower bound of the algorithm is $O(N\ Log\ N)$ hence our algorithm is optimal.

As illustrated earlier, the next best guest of any object o_u can be determined in $O(1)$. For each host o_u, the heap contains at most one pair (o_u, o_v). Hence, the maximum size of the heap is $O(N)$ which implies that each heap operation takes $O(Log\ N)$. In other words, a source incrementally returns the next best pair in $O(Log\ N)$.

7.4.2 External Memory Source

The basic idea of the external memory algorithm is the same as the internal memory algorithm. However, there are following two main challenges: 1) the heap cannot be stored in the internal memory and 2) finding the next best guest of an object requires accessing the sorted list of the objects which is stored in the external memory (this means that the algorithm would need to access the external memory every time the next best guest is to be determined).

We address the first challenge by using the external memory priority queue proposed by Arge [Arg03]. The basic idea of the external priority queue (or heap) is to retrieve and insert the elements in a batch which reduces the amortized I/O cost. Arge shows that the external priority queue can do an insert or delete operation in $O(\frac{1}{B}Log_{\frac{M}{B}}\frac{N}{B})$ amortized I/O where B is the number of elements that can be stored in one disk page,

$M \geq 2B$ is the number of elements that can be stored in the internal memory and N is the number of elements in the priority queue. For details, please see [Arg03].

We introduce the notion of dummy pairs to address the second challenge (i.e., to find the next best guest of an object without accessing the external memory). A dummy pair with a host o_u and a guest o_v is denoted as $(\overline{o_u, o_v})$. The pairs (o_u, o_v) we introduced earlier are called the regular pairs hereafter. Recall that when a regular pair (o_u, o_v) is retrieved from the heap, a pair $(o_u, o_{v'})$ is created and inserted in the heap where $o_{v'}$ is the next best guest of o_u. In contrast, when a dummy pair $(\overline{o_u, o_v})$ is retrieved from the heap, a dummy pair $(\overline{o_{u'}, o_v})$ is created and inserted in the heap where $o_{u'}$ is the next best host of o_v. The best host o_u is defined in a similar way as the best guest. More specifically, we say that an object o_u is a better host of o_v than $o_{u'}$ if $s(o_u, o_v) < s(o_{u'}, o_v)$. Finding the next best host is similar to finding the next best guest as described in previous section.

In Fig. 7.4, for each object, we show a regular pair with its best guest (curved arrows pointing right) and a dummy pair with its best host (connector style arrows pointing left). The scoring function is the sum of the attribute values and the score of each pair is shown on its edge.

Recall that when a pair (o_u, o_v) is retrieved from the heap, the next best guest $o_{v'}$ is determined by using the adjacent object information of o_v. For our external memory algorithm, we propose to store the adjacent object information with both the regular pairs and the dummy pairs. More specifically, with a regular pair (o_u, o_v), we attach the information of adjacent objects of the host o_u. In contrast, for a dummy pair $(\overline{o_u, o_v})$ we attach the information of adjacent objects of the guest o_v. The object that stores the adjacent object information in a pair is marked with a star. For example, $(\star o_u, o_v)$ denotes that the adjacent object information of o_u is attached with the pair (o_u, o_v).

Algorithm 16 presents the details of creating and maintaining an external memory source. The main idea behind the algorithm is that the heap is modified such that whenever a pair $(\star o_u, o_v)$ is retrieved from the heap, its dummy pair $(\overline{o_u, \star o_v})$ is the next best pair in the heap. These two pairs are retrieved and are used as follows; The next best

Algorithm 16 Creating and maintaining external memory source

```
1InitializeSource()
1: sort the objects in ascending order of their values
2: for each object $o_i$ do
3:     attach adjacent object's information with $o_i$
4:     $o_j \leftarrow$ the best guest of $o_i$
5:     $o_k \leftarrow$ the best host of $o_i$
6:     insert the pair $(\star o_i, o_j)$ into heap with score $s(o_i, o_j)$
7:     insert the dummy pair $(\overline{o_k, \star o_i})$ into heap with score $s(o_k, o_i)$

getNextBestPair()
1: get the top pair $(\star o_u, o_v)$ from the heap
2: get the next top pair (which is dummy pair $(\overline{o_u, \star o_v})$)/* Lemma 7.4.4 */
3: if next best guest of $o_u$ exists then
4:     $o_{v'} \leftarrow$ the next best guest of $o_u$
5:     insert the pair $(\star o_u, o_{v'})$ in heap with score $s(o_u, o_{v'})$
6: if next best host of $o_v$ exists then
7:     $o_{u'} \leftarrow$ the next best host of $o_v$
8:     insert the dummy pair $(\overline{o_{u'}, \star o_v})$ into heap with score $s(o_{u'}, o_v)$
9: return $(\star o_u, o_v)$
```

guest of o_u is determined by using the adjacent object information of o_v which is stored in the dummy pair $(\overline{o_u, \star o_v})$. Similarly, the next best host of o_v can be determined by using the adjacent object information of o_u which is stored in the regular pair $(\star o_u, o_v)$. It is easy to see that the next best pairs can be formed without accessing the external memory.

Modifying the heap priority function. It is important to clarify that if there exists more than one pair with the same score then the next best pair may not be the dummy pair of (o_u, o_v). Consider the example of Fig. 7.4, where several pairs have score 14. If

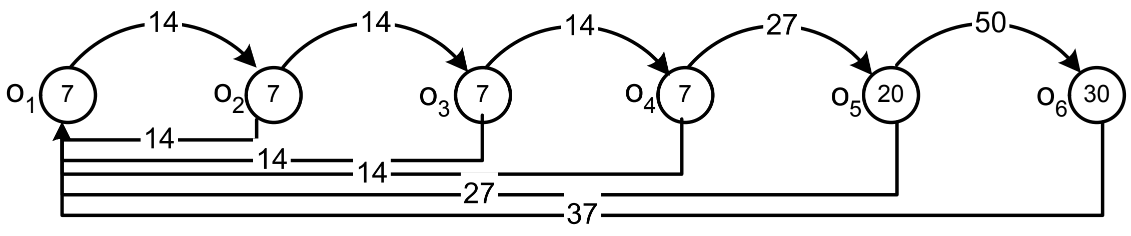


Figure 7.4: Illustration of dummy pairs

the heap accesses the pair (o_2, o_3), the next best pair cannot be its dummy pair $(\overline{o_2, o_3})$ because it has not been inserted in the heap yet. To guarantee that the next best pair is always the dummy pair of the retrieved pair, we modify the heap priority function as follows.

If two pairs have the same score, the heap gives priority based on the IDs of their guest objects. More specifically, if the scoring function is a right increasing function then the pair with the smaller ID of the guest object is given preference. If the scoring function is a right decreasing function then the pair with the larger ID of the guest object is given preference. The ID of each object in a source is its position in the list sorted in ascending order of attribute values. For instance, the ID of an object o_u is u.

If two pairs have the same score and the same guest object then the heap gives priority based on the IDs of their host objects. More specifically, if the scoring function is a left increasing function then the heap prefers the pair with the larger ID of the host object. If the function is a left decreasing function then the heap prefers the pair with the smaller ID of the host object.

If two pairs have the same score, the same guest object and the same host object then one of them is a regular pair and the other is its dummy pair. In this case, the heap gives priority to the regular pair.

Lemma 7.4.3 and Lemma 7.4.4 guarantee the correctness of the algorithm.

Lemma 7.4.3 *Given that the heap uses the priority function as described above. If a dummy pair* $(\overline{o_u, o_v})$ *is the top pair of the heap then its regular pair* (o_u, o_v) *has already been retrieved from the heap.*

Proof We prove the lemma for a function that is right increasing and left decreasing function. The proof for other functions is similar.

Assume that $(\overline{o_u, o_v})$ is the top pair. If the object o_u does not have any pair $(o_u, o_{v'})$ in the heap it implies that it has hosted o_v (i.e., (o_u, o_v) has been retrieved from the heap). If there exists a pair $(o_u, o_{v'})$ in the heap and $v' < v$, then $s(o_u, o_{v'}) \leq s(\overline{o_u, o_v})$ because the function is right increasing. This contradicts that $(\overline{o_u, o_v})$ is the top pair because the heap would prefer $(o_u, o_{v'})$ (even if the score $s(o_u, o_v) = s(o_u, o_{v'})$, the heap would prefer the pair $(o_u, o_{v'})$ because it has a guest with smaller ID).

If $v' = v$, this means that the regular pair (o_u, o_v) exists in the heap and a dummy pair cannot be the top pair in presence of its regular pair. If $v' > v$, this implies that the pair (o_u, o_v) has already been retrieved because, for a right increasing function, the pair of o_u with its guests that have smaller IDs are considered first. ■

Lemma 7.4.4 *Assume that the heap uses the priority function as described above. If a pair (o_u, o_v) is the top pair of the heap then its dummy pair $(\overline{o_u, o_v})$ is the second top pair.*

Proof We prove the lemma for a function that is right increasing and left decreasing function. The proof for other functions is similar.

We prove this by contradiction. Assume that $(\overline{o_u, o_v})$ is not present in the heap. At any stage, each host o_v contains a dummy pair $(\overline{o_{u'}, o_v})$ in the heap. If no such pair exists or $u' > u$ then this implies that the dummy pair $(\overline{o_u, o_v})$ has already been retrieved which violates Lemma 7.4.3. If $u' < u$ then $(\overline{o_{u'}, o_v})$ would be the top pair instead of (o_u, o_v). This is because $s(o_{u'}, o_v) \leq s(o_u, o_v)$ and the dummy pair $(\overline{o_{u'}, o_v})$ would be preferred even when $s(o_{u'}, o_v) = s(o_u, o_v)$ because the heap prefers the pair with smaller host ID if the score and the guest objects are same. ■

Please note that once the source is created, it does not require to access the external memory to create new pairs. The only external memory I/Os are due to insertion and deletion from the external memory heap. The cost of returning the first pair is sorting the objects and inserting $O(N)$ pairs in the external heap. Hence, the cost is $O(\frac{N}{B}Log_{\frac{M}{B}}\frac{N}{B})$ which is I/O equivalent to $O(N\ Log\ N)$ internal memory algorithm and hence is optimal [Vit01]. The amortized I/O cost for retrieving next best pair is $O(Log_{\frac{M}{B}}\frac{N}{B})$ which is I/O equivalent to the cost of internal memory source.

7.5 Query Processing Algorithm

7.5.1 Technique

We apply the threshold algorithm (TA) [FLN03, NR99, GBK00] to combine the scores of a pair from different sources and return the top-k pairs. However, please note that TA assumes that the sources support the random accesses (see Section 7.2.2). In other words, when a pair is returned from a source S_i, TA needs to obtain its score on every other attribute. We enable TA to access the scores of a pair on other attributes as follows.

For internal memory algorithms, we assume that the objects are stored in the main memory (this consumes $O(dN)$ memory space). When a pair (o_u, o_v) is returned from one of the sources, we use the object table and retrieve the attribute values of o_u and o_v and compute the score of (o_u, o_v) on every other attribute.

For the external memory algorithm, doing the random access requires accessing the object table (which exists in the external memory). This would be quite expensive because we need to look up the attribute values of two objects for each seen pair and this may require two I/Os. One solution is to apply NRA algorithm [FLN03] because it does not require random accesses. However, NRA algorithm requires an unbounded buffer (see Section 7.2.2) and the main memory consumption may be prohibitively large (it may be $O(V)$ where V is the total number of valid pairs).

To enable random accesses for TA, we modify each source S_i such that each pair stores d attribute values of both of the objects. This increases the amortized I/O cost of creating the external memory source by a factor d because the number of entries that can be stored in one disk block is reduced. However, doing this allows us to compute the score of each pair on every attribute without any additional I/O. Although this approach may increase the disk usage, the external memory sources are required only during the query processing and the data can be deleted after the query has been answered.

Incrementally returning top-k pairs. Incremental algorithms return the results one-by-one, i.e., the partial results are incrementally reported to the users without waiting for all the results to be computed. We show that we can modify TA such that it reports top-k pairs incrementally. For instance, the best pair is reported to the user before the computation of remaining top-(k-1) pair is completed. In some applications, incremental algorithms are preferred [HS98] because i) the query engine can use the algorithm in pipelined fashion and ii) such algorithms aim to report the results as soon as possible and the user may terminate the algorithm if he is happy with top-m (for $m < k$) results returned so far. In Section 7.6.3, we demonstrate the importance of incrementally reporting the results where we use this feature to optimize the performance of our algorithm for answering exclusive top-k pairs queries.

Recall that TA maintains W_k which corresponds to the largest of the scores of the objects maintained in the heap (see Section 7.2.2). The algorithm terminates when the threshold t becomes at least equal to W_k. Let W_i be the i-th largest score of the objects maintained in the heap. It can be easily proved that when $t \geq W_i$, the object with i-th largest score in the heap is among top-i pairs. We modify TA such that it starts by setting $i = 1$ and reports the best pair as soon as $t \geq W_1$. The algorithm continues by iteratively incrementing i by one and reporting the i-th best pair as soon as $t \geq W_i$. It can be easily shown that the complexity of this variation of TA is the same as that of the original TA.

7.5.2 Complexity Analysis

The number of elements accessed by TA is always less than or equal to the number of elements accessed by Fagin's Algorithm (FA) [FLN03] (see Section 7.2.2). FA algorithm stops the sorted accesses when exactly k elements are returned from all d sources. Let

V be the number of elements in each source. The expected number of sorted accesses by FA is $T = O(V^{(d-1)/d}k^{1/d})$ under the assumption that the score of an element in one source is independent of its score in other sources [Fag99].

As the cost of TA is always less than or equal to FA, the number of pairs our algorithm is expected to access from each source is $O(T)$ assuming that the score of a pair in one source is independent of its score in the other sources. The total number of accesses from all d sources is $O(dT)$. As shown earlier, the cost of accessing a pair from a source is $O(Log\ N)$, hence the total expected cost [4] for the internal memory algorithm is given by Eq. (7.1).

$$O(dT\ Log\ N) = O(d\ V^{\frac{d-1}{d}} k^{\frac{1}{d}} Log\ N) \tag{7.1}$$

For the non-chromatic queries, the total number of valid pairs $O(V)$ is $O(N^2)$. Hence the expected cost of our algorithm to answer a two dimensional closest pair query is $O(N\ Log\ N)$ which is optimal in algebraic decision tree model [BO83].

The cost of our external memory algorithm can be obtained similarly. The amortized I/O cost of accessing dT (T pairs from each source) is $O(\frac{dT}{B}(Log_{\frac{M}{B}} \frac{N}{B}))$ where B is the number of pairs that can be stored in one block and $M \geq 2B$ is the number of pairs that can be stored in the main memory reserved for an external priority queue. For a two dimensional non-chromatic closest pair queries, the expected amortized I/O cost is $O(\frac{N}{B}(Log_{\frac{M}{B}} \frac{N}{B}))$ which is I/O equivalent to $O(N\ Log\ N)$ internal memory algorithm hence is optimal [Vit01].

The space usage of the internal memory algorithm is $O(dN)$ because the main algorithm stores a table containing N objects with d attributes for each object and each source stores a table of N objects with one attribute value for each object. The main memory usage of the external memory algorithm is $O(k + dM)$ where M is the memory used for each source. The minimum memory an external source requires is $2B$, hence the

[4]Note that the cost analysis includes the cost of creating the sources. The cost of creating d sources is $O(d(N\ Log\ N))$ which is dominated by Eq. (7.1). Same holds for the cost analysis of the external memory algorithm. Our experiment results also include the cost of creating the sources.

minimum main memory requirement is $O(k + 2dB)$.

7.6 Extensions

7.6.1 Skyline Pairs Query

A pair (x, y) is said to *dominate* another pair (a, b) if for every attribute i, $s_i(x, y) \leq s_i(a, b)$ and for at least one attribute j, $s_j(x, y) < s_j(a, b)$. A skyline pairs query returns every pair that is not dominated by any other pair.

It can be shown that for any monotonic global scoring function, the best pair is always one of the skyline pairs. In other words, the skyline pairs query gives shortlisted candidate pairs such that for every candidate pair there exists a global scoring function for which it is the best pair. Hence if the users cannot define a suitable scoring function, they can select a pair from the skyline pairs that best meets their requirement.

Consider the example of a person who is interested in buying a broadband internet connection and a home phone connection. He might want to retrieve the pairs (broadband and phone) that have low total monthly cost, low total setup fee and shorter average contract length. Suppose that a database stores the information of broadband and home phones provided by different companies. While the score-based top-k pairs queries can be used to retrieve the top-k pairs, the user may instead prefer to retrieve all the pairs that are not dominated by any other pair (i.e., return every pair such that no other pair has lower total monthly cost, lower total setup fee and shorter average contract length).

Technique

For the ease of the presentation, we assume that all the pairs in a source have unique scores. Later, we will present the approach to handle the case when more than one pair can have same score. Our algorithm is similar to Fagin's Algorithm (FA) (see Section 7.2.2). However, unlike FA algorithm, we address the problem of unbounded buffer. Our algorithm works as follows.

1 . Do the sorted accesses on each source S_i. For each newly seen pair p, determine its score on all other attributes. Compare p with existing skyline pairs and include it in the set of skyline pairs if it is not dominated by any existing skyline pair. Otherwise, discard it.

2. Terminate when at least one object has been seen under the sorted accesses from all the sources. Report the skyline pairs.

The correctness of the algorithm follows from the fact that a pair p cannot be dominated by any pair p' that is accessed after it. This is because the score of p' is larger than p in at least one source. The termination condition is also correct because if a pair p is seen in every source then every pair p' that has not been seen in *any* source is dominated by p.

If more than one pairs have same score in a source S_i then a pair p can be dominated by a pair p' that is accessed after it. This is because p' may have a score equal to p in the source S_i and may have smaller scores in all other sources. We address this issue as follows. Let x_i be the score of a pair p that has been accessed from a source S_i. We discard the pair p if it is dominated by any of the existing skyline pairs. Otherwise, we insert it in a list C which contains the candidate skyline pairs. When a pair p' is accessed from S_i, if its score is equal to x_i it is compared with every pair in C and the pairs that are dominated by p' are deleted. Whenever the score of p' is larger than x_i, all the pairs in C are confirmed as the skyline pairs and are inserted in the set of skyline pairs.

Let $score_i$ be the score of a pair p in a source S_i such that p has been seen under sorted accesses on all sources. The algorithm terminates if the score x_i of the last pair seen in a source S_i is larger than $score_i$. This is because every unseen pair has a score on S_i larger than that of p and cannot have score less than the score of p on every other source. The proof of correctness is straight forward and is omitted. Note that the algorithm is incremental, i.e., it can report the skyline pairs without computing all of the skyline pairs.

A k-skyband [PTFS05] query returns every element that is dominated by at most

$(k-1)$ other elements. A k-dominant skyline [CJT+06] query returns every element that is not dominated by any other element in k or more dimensions. We remark that the extension of the algorithm to answer k-skyband pairs query and k-dominant skyline pairs query is straight forward.

Analysis

We assume that the pairs have unique scores in each source. The number of accesses from each source is equal to the accesses by FA (because the algorithm stops when at least one object has been returned from all sources). So, the expected number of accesses from each source is $T = O(V^{(d-1)/d})$ (the value of k is one). The expected number of total accesses on all the sources is $O(dT)$.

For each retrieved pair, we compare it with all the existing skyline pairs. The average number of skyline pairs is estimated to be $O(Log^{d-1}V)$ [BKST78]. Since V is at most $O(N^2)$, the expected number of skyline pairs is $O(Log^{d-1}N)$. Hence the expected cost of the internal memory skyline pairs algorithm is $O(dT\ Log^{d-1}N)$. The expected amortized I/O cost is the same as the cost of score-based top-k $(k = 1)$ pairs query obtained because the cost was obtained using the number of accesses by FA.

The lower bound on the cost of the skyline pairs queries is $O(N\ Log\ N)$. This is because the lower bound cost of the closest pairs query is $O(N\ Log\ N)$ and a closest pairs query can be solved by scanning the skyline pairs once. It is easy to see that the expected cost of our algorithms meets the lower bound of the skyline pairs queries if two or less attributes are involved. The expected main memory usage of the internal memory algorithm is $O(dN + Log^{d-1}N)$ because in addition to the object table, it also stores the existing skyline pairs. The expected main memory requirement of the external memory algorithm is $O(k + 2dB + Log^{d-1}N)$.

7.6.2 Rank-based Top-k Pairs Queries

In order to define a suitable scoring function, the users must have sufficient domain knowledge. Moreover, it is difficult to define a global scoring function on the attributes that are incompatible (e.g., dollars and inches) [FKS03]. In such cases, the users can issue a rank-based top-k pairs query defined below.

First, we define the rank of a pair (a, b) on an attribute i denoted by $rank_i(a, b)$. Let s_i be the loose monotonic scoring function for the attribute i. $rank_i(a, b)$ is the number of pairs (x, y) for which $s_i(x, y) < s_i(a, b)$. In other words, if the pairs are sorted in ascending order of their scores on i^{th} attribute, $rank_i(a, b)$ is the rank of the pair (a, b) in the sorted order.

Given a global scoring function f, the final rank-based score R_SCORE of a pair (a, b) is;

$$R\_SCORE(a, b) = f(rank_1(a, b), \cdots, rank_d(a, b)) \tag{7.2}$$

Given a set of objects O, a rank-based top-k pairs query returns a set of pairs $P \subseteq O \times O$ that contains k pairs such that for any pair $(a, b) \in P$ and any pair $(a', b') \notin P$, $R\_SCORE(a, b) \leq R\_SCORE(a', b')$.

Technique

When a pair p is seen on a source S_i, although its score on the other sources can be determined, it might not be possible to determine its rank on the other sources. In other words, the random access on a source cannot determine the rank of a pair in this source. However, if a pair p is seen under the sorted access then its rank is the number of pairs that have been returned by this source and have smaller scores. This can be easily done by maintaining a counter for each source. The problem of rank-based top-k pairs can be solved by using NRA [FLN03] because the sorted accesses are possible but the random accesses are not possible.

As mentioned in Section 7.2.2, there are two major weaknesses of NRA. First is that whenever a new element is seen under the sorted access, the best possible scores of all the previously seen pairs are to be updated. This problem has been addressed by LARA algorithm [MYCC07] which we briefly described in Section 7.2.2. The second problem is that NRA uses an unbounded buffer. We reduce its memory usage by the following observation. A pair p that is dominated by k other pairs cannot be the top-k pair. Hence, we only need to maintain the $(k+1)$-skyband pairs. Other pairs can be safely pruned.

We remark that, by using the strategy presented in Section 7.5.1, NRA can be modified to report the top-k pairs in an incremental fashion.

Analysis

In the worst case, the growing phase of LARA (see Section 7.2.2) completes when there are at least k elements that are seen on all the sources. Hence, the expected number of pairs accessed from each source is at most equal to the number of pairs accessed by FA. So, the expected number of pairs accessed from each source during the growing phase is $T = O(V^{(d-1)/d} \cdot k^{1/d})$. In the growing phase, when a pair p is retrieved, it is compared against all $(k+1)$-skyband pairs to see if it can be pruned. The expected size of $(k+1)$-skyband is $O(k\ Log^{d-1}N)$ [ZLZ+09]. So, the expected cost of the growing phase is $O(dkTLog^{d-1}N)$ because in total dT pairs are accessed and each pair is compared with every pair in the $(k+1)$-skyband.

Now, we estimate the number of elements accessed by the shrinking phase of LARA (which cannot be more than the number of elements accessed by NRA). We assume that the global function is sum of the local scores. Moreover, we assume that the score of every pair in a source is unique. As stated earlier, when $O(T)$ elements are accessed from each source, the algorithm is expected to see k elements that have been returned by all the sources. The worst possible score of these k elements is $W_k = dT$ (the rank of the pair is T in each source). If dT elements are accessed from each source, then the algorithm can stop. This is because the final score of every object that is not seen in at least one source cannot be smaller than $W_k = dT$. Hence, the number of accesses by NRA on each source is at most dT where $T = O(V^{(d-1)/d} \cdot k^{1/d})$. The total number of accesses on all d sources is $O(d^2T)$. The cost of each access in the shrinking phase is $O(Logk + 2^d)$ [MYCC07]. Hence the expected total cost of the shrinking phase is $O(d^2T(Log\ N + Log\ k + 2^d))$.

The total cost of the internal memory rank-based top-k pairs query is the sum of the cost of the growing phase and the cost of the shrinking phase as computed above. The expected amortized I/O cost of the external memory algorithm is $O(\frac{d^2T}{B} Log_{\frac{M}{B}} \frac{N}{B})$ because d^2T pairs are expected to be accessed from the sources.

The expected main memory requirement for the internal memory algorithm is $O(dN + k\ Log^{d-1}\ N)$ because the pairs in $(k+1)$-skyband are also kept in the memory. The expected main memory requirement of the external memory algorithm is $O(2dB + k\ Log^{d-1}\ N)$.

7.6.3 Exclusive Top-k Pairs Queries

Consider a set S containing top-k pairs. We say that an object o_u satisfies the exclusiveness constraint if o_u appears *at most* once in the set of pairs S, i.e., there exists at most one pair related to o_u in S.

An exclusive top-k pairs (ETP) query retrieves k pairs with the smallest scores such that every object o_u satisfies the exclusiveness constraint. In such queries, if the best pair returned by the query is (o_u, o_v) then the algorithm continues retrieving the remaining top-$(k-1)$ pairs by ignoring the objects o_u and o_v and the pairs involving these two objects. The traditional top-k pairs queries studied earlier in this chapter are called inclusive top-k pairs queries throughout this section.

Exclusive k closest pairs query [UMY07] is a special case of exclusive top-k pairs queries and it uses Euclidean distance as the scoring function. An ETP query has many interesting applications. For example, an exclusive k closest pairs query can be issued to solve car-parking assignment problem [UMY07] where each parking slot is to be reserved for *at most* one car (the car that is closest to it). Consider another example of a recruitment agent who has a list of applicants and a list of jobs. On a given day, he may want to arrange k interviews (one interview corresponds to one job-applicant pair). He may issue an exclusive top-k pairs query to retrieve k job-applicant pairs such that the scores (suitability) of the reported job-applicant pairs are better than the remaining pairs. Due to time constraints, it may not be possible for an applicant to appear in more than one interviews on a given day. Hence, the agent may issue the query such that every applicant satisfies the exclusiveness constraint.

In the above example, the agent may have no problem arranging more than one interviews for a single job in a single day. Hence, he may not require that every job also satisfies the exclusiveness constraint. Although, for the sake of simplicity, we focus to present the techniques for the case where every object is required to satisfy the exclusiveness constraint, we remark that our techniques can be easily extended to answer the queries where only a certain type of objects (e.g., applicants) are required to satisfy the exclusiveness constraints. Also, our techniques can be easily applied to solve more general exclusiveness constraint. A more general exclusiveness constraint requires that an object o_u appears *at most* m times in the top-k pairs.

We also remark that our techniques can be easily extended to answer rank-based exclusive top-k pairs queries. However, in this section, we focus only on the score-based exclusive top-k pairs queries.

Technique

As described in Section 7.2.2, the threshold algorithm (TA) stops when the threshold t is at least equal to W_k where W_k is the score of k-th best pair seen so far. In other words, W_k is the worst score of the current top-k pairs maintained in a heap. Note that TA cannot be directly applied for ETP queries because the heap may contain more than one pairs involving the same object. Hence, W_k may not correspond to the score of k-th exclusive best pair.

To answer ETP queries, TA needs to be modified such that every object in the heap satisfies the exclusiveness constraint (i.e., for each object o_u, there exists at most one pair in the heap that involves o_u). After this modification, W_k corresponds to the score of current k-th best exclusive pair. This guarantees that the stopping condition is correct.

The heap can be easily modified to ensure that every object in the heap satisfies the exclusiveness constraint . We do this as following. Assume that a new pair p is retrieved by TA and $p.score$ is its score. If $p.score$ is greater than W_k then p is discarded because it cannot be one of the top-k exclusive pairs. Otherwise, we check if there exists a pair p' in the heap such that both p and p' have one common object. If $p.score$ is smaller than $p'.score$ then p' is deleted from the heap and p is inserted in the heap. This is because p' cannot be one of the top-k exclusive pairs in the presence of p. Otherwise, if $p.score$ is larger than $p'.score$ then p is discarded.

Optimizations.

Assume that TA is modified in the way as described in Section 7.5.1 such that it incrementally reports the top-k exclusive pairs. If the algorithm reports a pair $p = (o_u, o_v)$ as one of the top-k exclusive pairs, we should modify the sources such that they ignore every pair that contains either o_u or o_v. To do this, we can delete o_u and o_v from the list of objects so that the corresponding pairs are not considered by the source. This improves the efficiency of the sources because the sources are required to handle fewer objects and hence fewer pairs.

Next, we show how to efficiently delete an object o_u and all the pairs related to it from a source S_i. When an object o_u is deleted, the following three steps should be taken to guarantee the correctness of Algorithm 15 (the readers are encouraged to refresh their memories by going through Algorithm 15).

Step 1. Delete from the heap every pair that contains o_u.

Step 2. For each deleted pair (o_x, o_u), insert (o_x, o_w) in the heap where o_w is the next best guest of o_x. This step guarantees that the heap contains, for each host o_x, a pair formed with its best guest among the guests that it has not hosted earlier.

Step 3. Adjust the left and right adjacent objects after the deletion of o_u.

First, we present the techniques to efficiently implement the step 3. Then, we present the techniques to efficiently implement steps 1 and 2.

Implementation of Step 3. Without loss of generality, we show the technique of how to adjust the pointers of right adjacent objects after an object o_u is deleted from a source. The technique for adjusting the pointers for left adjacent objects is similar. Fig. 7.5 shows an example where only the pointers for right adjacent objects are shown. Adjusting the pointers for non-chromatic queries is the same as adjusting the pointers for a linked list. Hence, we omit the details.

For heterochromatic queries, we adjust the pointer as follows. Assume that the object o_u is deleted. We need to adjust any pointer that points to o_u. In other words, if o_u was a right adjacent object of an object o_x, we need to determine the new right adjacent object of o_x. If the object o_{u+1} (the object next to o_u) has a color different than o_x then the right adjacent object of o_x is o_{u+1}. Otherwise, if o_x and o_{u+1} have the same color then the right adjacent object of o_{u+1} is set as the right adjacent object of o_x because it is guaranteed to have a color different from o_x.

Consider the example of Fig. 7.5(b) and assume that the deleted object is o_3. Since o_3 is the right adjacent object of o_2, we need to adjust the right adjacent object of o_2 after o_3 is deleted. Since o_4 (the next object of o_3) has the same color as of o_2, o_4 is not the right adjacent object of o_2. Hence, o_6 which is the right adjacent object of o_4 is set as the right adjacent object of o_2.

For homochromatic queries, we adjust the pointers as follows. Assume that o_u is the deleted object and we need to adjust the right adjacent object of o_x. The right adjacent object of o_x is set as the right adjacent object of o_u. Consider the example of Fig. 7.5(c) and assume that o_3 is deleted. The right adjacent object of o_1 is to be adjusted. The new right adjacent object of o_1 is o_6 which is the right adjacent object of o_3.

Implementation of Step 1 and Step 2. First, we show a straight forward approach to implement the step 1 and step 2. Then, we show a better approach. A straight forward approach to delete all the pairs related to o_u is to maintain the pointers to each pair in

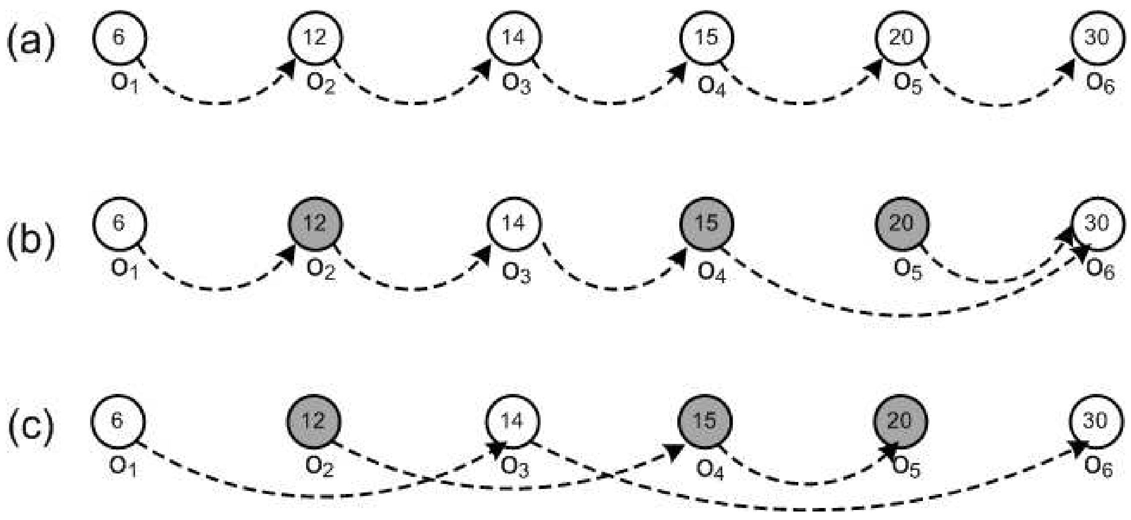


Figure 7.5: (a) Non-chromatic (b) Heterochromatic (c) Homochromatic

the heap that is related to o_u. When o_u is deleted, these pointers may be used to delete the related pairs. When a pair (o_x, o_u) is being deleted, the next best guest o_w of o_x is determined in the way as explained in Section 7.4.1 and the new pair (o_x, o_w) is inserted in the heap.

Next, we show a more efficient approach. Consider the example of Fig. 7.6(a). Recall from the description of Example 7.4.1 that all the pairs connected by the arrows are the pairs that have been inserted in the heap. More specifically, the pairs (o_1, o_2), (o_2, o_5), (o_3, o_5), (o_4, o_5) and (o_5, o_6) are inserted in the heap with scores 6, 8, 6, 5 and 10, respectively. Now assume that the object o_5 is deleted. The algorithm needs to delete the pairs (o_2, o_5), (o_3, o_5), (o_4, o_5) and (o_5, o_6) from the heap. Note that each deletion from the heap takes $O(log\ N)$ where N is the number of pairs in the heap. Next, we present a strategy that guarantees that each object has at most two pairs related to it in the heap.

Recall that Algorithm 15 ensures that each object o_u can have at most one pair (o_u, o_v) in the heap such that o_u is the host object. For instance, there is only one pair in the heap (o_5, o_6) for which o_5 is the host object (see the outgoing edge of o_5 in Fig. 7.6(a)). However, there may be more than one pairs in the heap for which o_u is the guest object. For example, there are three pairs (o_2, o_5), (o_3, o_5), (o_4, o_5) for which o_5 is the guest object

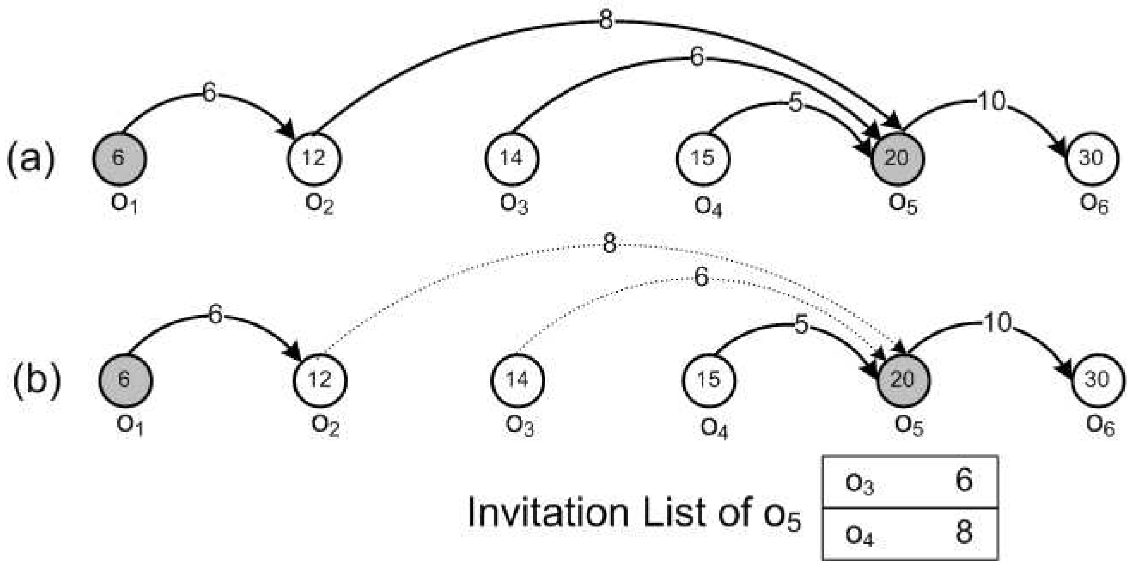


Figure 7.6: Invitation list of o_5

(see the incoming edges of o_5 in Fig. 7.6(a)).

We modify Algorithm 15 such that each object o_u has at most one pair in the heap for which o_u is the host object. In other words, even if there are more than one incoming edges of o_u, only at most one incoming edge is inserted in the heap (the one with the smallest score). In the example of Fig. 7.6(b), the pair (o_4, o_5) is inserted in the heap. The other incoming edges of o_5 (shown in broken lines) are not inserted in the heap. Instead, they are inserted in a list called the *invitation list* of o_5 as shown in Fig. 7.6(b). The invitation list of an object o_5 records the objects that will host o_5 in future. For instance, when the pair (o_4, o_5) is reported to the main algorithm, the invitation list is used to insert a new pair (o_3, o_5) in the heap.

The invitation list of each object o_u is always kept sorted so that the next pair that is to be inserted in the heap can be determined efficiently. Note that the cost of keeping the invitation list sorted is $O(log\ m)$ per insertion where m is the size of the invitation list. Note that if the invitation list is not maintained and all these pairs are inserted in the heap (as done in Algorithm 15), the cost would be higher. This is because the cost of inserting a pair in the heap is $O(log\ N)$ which is higher than $O(log\ m)$ because $N > m$.

When an object o_u is deleted, for each object o_x in its invitation list, the next best guest o_w is determined and the object o_x is inserted in the invitation list of o_w.

Analysis

We provide the analysis for the simpler version of our algorithm and the cost of the optimized algorithm is always at most equal to the cost of the simpler version. Assume that the algorithm terminates after accessing Z elements from each source. Let M be the average number of valid pairs containing any object o_u. If a pair (o_u, o_v) is reported to the user, TA ignores at most $2M$ unseen pairs (M pairs containing o_u and M pairs containing o_v). If k pairs are reported, the number of ignored pairs is at most $2kM$. Assuming that the algorithm terminates after accessing Z pairs from each source, the number of pairs that the algorithm ignores is at most $\frac{Z \cdot 2kM}{V}$ where V is the total number of valid pairs. The algorithm stops when $T = O(V^{(d-1)/d} \cdot k^{1/d})$ have been seen from each source [Fag99] excluding the pairs that have been ignored.

$$Z - \frac{Z \cdot 2kM}{V} = V^{(d-1)/d} \cdot k^{1/d} \tag{7.3}$$

The above equation can be solved to compute the value of Z. Note that Z is at most equal to V (the total number of valid pairs).

$$Z = min(V, \frac{V}{V - 2kM} \cdot V^{(d-1)/d} \cdot k^{1/d}) \tag{7.4}$$

For non-chromatic queries, V is at most N^2 and M is equal to N. Hence, the number of elements accessed from each source is equal to $min(N^2, \frac{N}{N-2k} \cdot N^{2(d-1)/d} \cdot k^{1/d})$. Note that this number is $N/(N-2k)$ times the number of pairs accessed for the score-based top-k pairs queries studied in Section 7.5. Hence, for the queries where $k << N$ (which is the case in many real world applications), the cost of our algorithm to answer exclusive top-k pairs queries is close to the cost of our algorithm to answer inclusive top-k pairs queries.

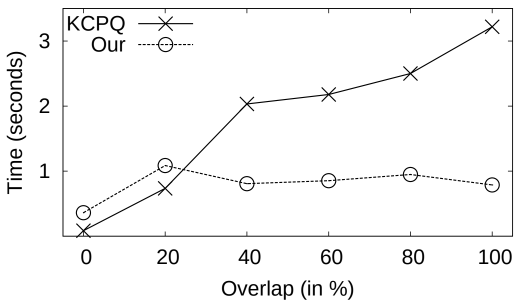
7.7 Experiments

We conducted extensive experiments on both real and synthetic data sets. We show that our algorithm for the score-based top-k pairs queries outperforms the existing best known algorithm (KCPQ) [CMTV00] for the k-closest pairs queries. For the queries that use more generic functions, we compare our algorithm with a naïve algorithm because there does not exist any other work to handle such queries.

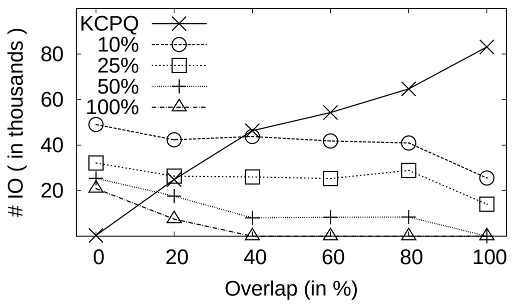
7.7.1 k-Closest Pairs Queries

We compare our algorithm with the best known k-closest pairs algorithm called KCPQ [CMTV00]. In accordance with [CMTV00], the page size for both of the algorithms is set to $1K$. The k closest pairs query joins two data sets each containing $100,000$ objects and returns the k closest pairs. k is set to 10 in all experiments unless mentioned otherwise.

It has been noted that the overlap between the data sets is one of the main factors [CMTV00] that affect the performance of the existing algorithms. Fig. 7.7 shows the effect of the overlap on KCPQ and our algorithm. In Fig. 7.7(a), we run both of the algorithms in the internal memory and observe that our algorithm is 2 to 3 times faster when the overlap is at least 40%. For the smaller overlaps, the performance of KCPQ is better because most of the intermediate nodes of the R-trees are quickly pruned. However, its performance is still not significantly better than our algorithm. Note that our algorithm is not sensitive to the data overlap.



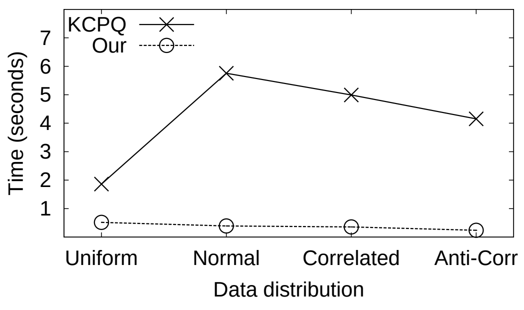
(a) Internal Memory



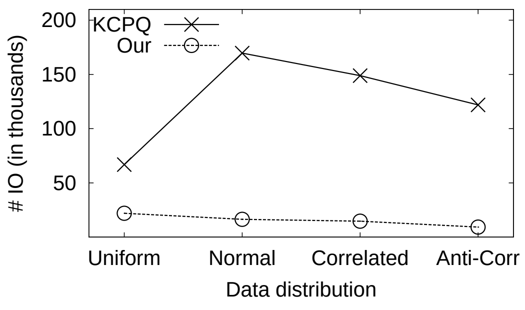
(b) External Memory

Figure 7.7: Effect of overlapping

Fig. 7.7(b) shows the performance of both of the algorithms in the external memory. The heap of KCPQ algorithm contains the intermediate nodes of the R-trees. Consequently, it uses larger amount of main memory. The buffer size for our algorithm is set according to the main memory usage of KCPQ. More specifically, we run our algorithm with the buffer size set to 100%, 50%, 25% and 10% of the memory used by KCPQ. Fig. 7.7(b) demonstrates that when the overlap is 40% or more, our algorithm performs better even when the memory used by our algorithm is 10% of the memory used by KCPQ.



(a) Internal memory



(b) External memory

Figure 7.8: Different data distributions

We also conducted several experiments on different data distributions. More specifically, we generated the data sets following uniform, normal, correlated and anti-correlated distributions. For each distribution, we generated two data sets with 50% overlap between them. Fig. 7.8 demonstrates that our algorithm is not affected by the data distribution and performs significantly better than KCPQ.

We compared the two algorithms for several other parameters and data sets and observed that although our algorithm supports more general scoring functions and does not require pre-built indexes, it outperforms KCPQ for all settings except when the overlap is too small.

7.7.2 Queries Involving Generic Scoring Functions

For the general scoring functions, we compare our algorithms with a naïve algorithm. The naïve algorithm uses nested loop to join a data set with itself (block nested loop for external memory processing). The disk page size is set to $4K$. The buffer size for each of our external memory source is set to 2 pages (this is the minimum required by the external priority queue [Arg03]).

Real Data

The real data set[5] consists of location data consisting of $304,895$ location points belonging to 87 zip codes of USA. The zip codes roughly map to different towns (or suburbs). Each point in the data set corresponds to a residential block. We extracted the coordinates of the streets and the number of addresses along each street. We treat the center of each street as a residential block and the number of addresses along the street as the population of the block. For each block, we randomly generate a value which denotes the average rent of the houses in the residential block. All of the attributes are normalized to a unit space. The global scoring function we used is the sum of the local scores.

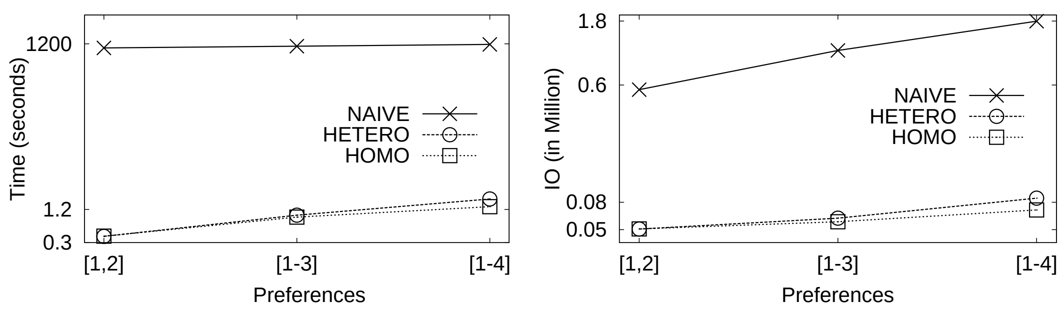
Table 7.1: The queries used on real data

Preference	Heterochromatic	Homochromatic
1&2: Distance	close	far
3: Population	high	high
4: Rent	low	low

We use several heterochromatic and homochromatic queries each involving two to four attributes. Table 7.1 shows some of the queries we use on the real data. First two preferences involve two attributes (i.e., the two location coordinates of each block). A heterochromatic query on these two attributes retrieve the closest pairs of blocks such that each block is located in a different suburb. For a query involving d preferences, we use the first d preferences for that query listed in the table. For example a homochromatic query on three attributes retrieves the pairs of blocks (located in the same suburb) that are far from each other and have high total population. k is set to 10 for all queries.

Fig. 7.9 shows that the naïve algorithm is three orders of magnitude slower than our internal memory algorithm and uses an order of magnitude more IOs. The query time

[5]http://www.census.gov/geo/www/tiger/



(a) Internal Memory (b) External Memory

Figure 7.9: Real data

for our algorithm is low which demonstrates the applicability of our approach in the real world applications. Similar results were observed when the queries were run for other parameters.

Synthetic Data

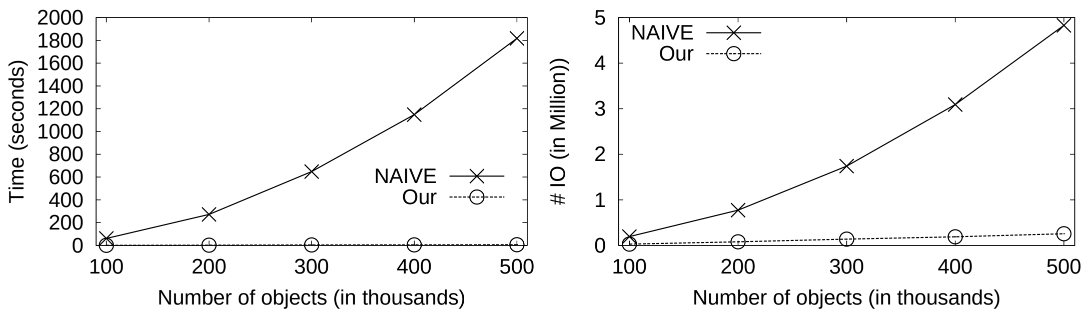
The default synthetic data set contains the points following a uniform distribution. Each object is randomly assigned a color. The number of colors vary from 50 to 250. The local scoring functions used by the algorithms are the sum and the absolute difference. The global scoring function is a weighted aggregate (we allow negative weights). For each dimension, a local scoring function is randomly chosen (sum or absolute difference) and is assigned a random weight.

Table 7.2: Experiment parameters

Parameter	**Range**
Number of objects ($\times$1000)	100, 200, **300**, 400, 500
Number of colors	50, **100**, 150, 200, 250
Number of attributes	2, 3, **4**, 5, 6
k	1, **10**, 25, 50, 100

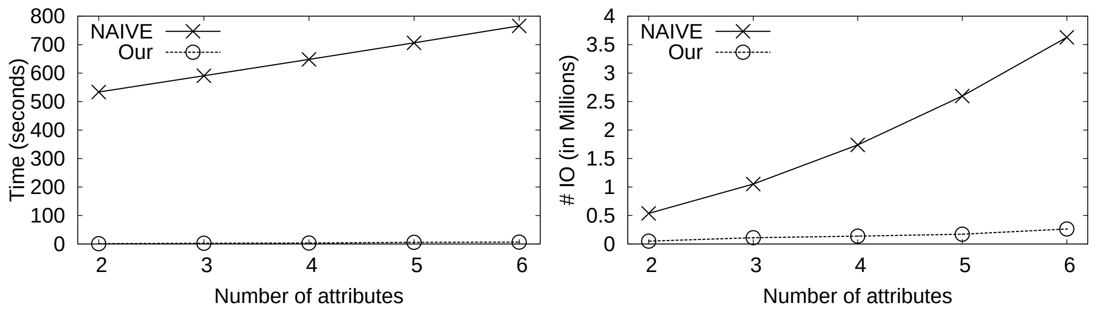
We present the results for the homochromatic top-k queries. The results for the non-chromatic and the heterochromatic queries follow similar trends. Table 7.2 shows the default parameters in bold.

Fig. 7.10 and Fig. 7.11 study the effect of increasing the number of objects and the number of attributes, respectively. While the performance of both of the algorithms degrades, our algorithm scales very well.



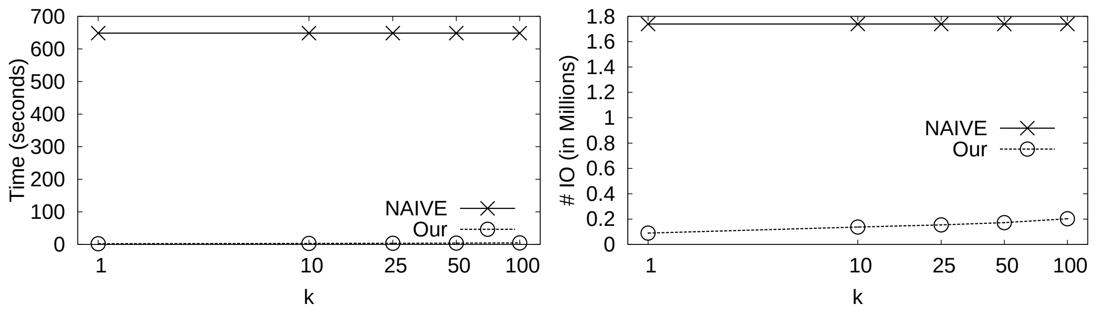
(a) Internal Memory (b) External Memory

Figure 7.10: Effect of number of objects



(a) Internal Memory (b) External Memory

Figure 7.11: Effect of number of attributes



(a) Internal Memory (b) External Memory

Figure 7.12: Effect of k

Fig. 7.12 studies the effect of k. The performance of our algorithm is better for smaller k. The naïve algorithm is not affected by k because it considers all the pairs regardless of the value of k.

Fig. 7.13 studies the effect of number of colors. Our algorithm performs slightly better when the number of colors is large. This is mainly because the number of valid pairs decreases when the number of colors is large. However, the effect is not very significant because the number of pairs that are accessed from each source is not significantly affected.

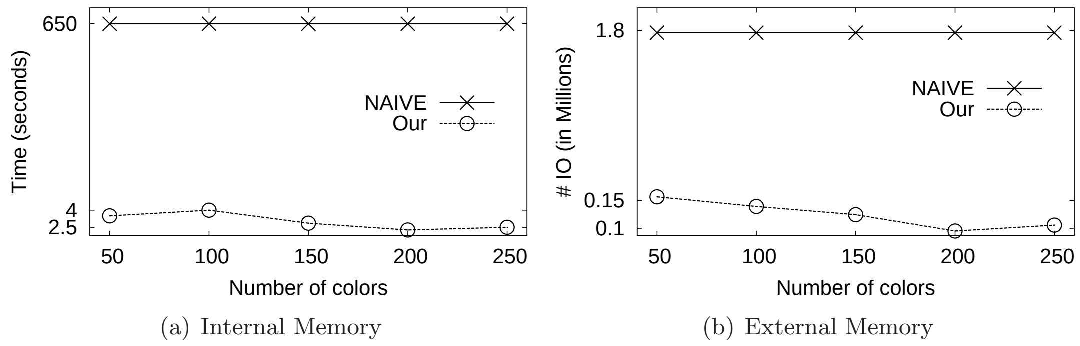


Figure 7.13: Effect of number of colors

Finally, we present the results for the skyline pairs queries and the rank-based top-k pairs queries. As stated earlier, the naïve algorithms perform extremely bad. Therefore, we compare the performance of our proposed algorithms (for the score-based queries, the rank-based queries and the skyline queries) to give the readers an insight about the cost of each type of query.

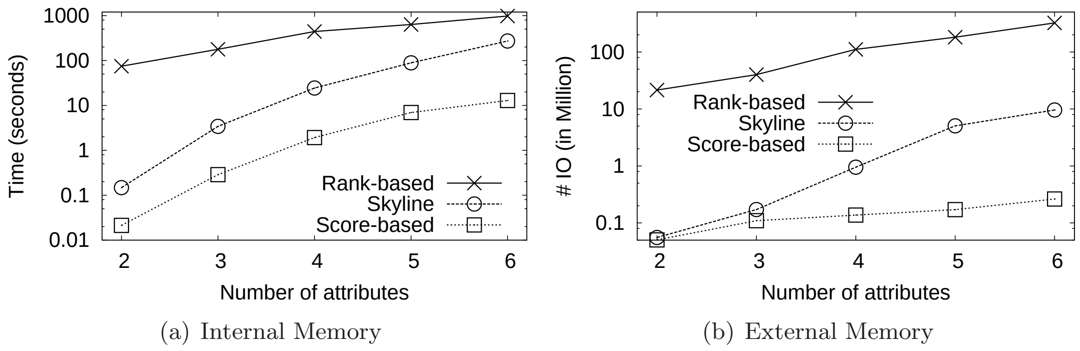


Figure 7.14: Comparison of different top pairs queries

Fig. 7.14 shows the cost of the three algorithms when different number of attributes are involved in the query. The score-based top-k queries are the easiest to solve among the three and the rank-based top-k pairs queries are the hardest. The cost of each of the algorithms increases with the number of attributes used in the query.

7.8 Summary

Top-k pairs queries have many real applications. k closest pairs queries, k furthest pairs queries and their bichromatic variants are some of the examples of the top-k pairs queries that rank the pairs on distance functions. While these queries have received significant research attention, there does not exist a unified approach that can efficiently answer all these queries. Moreover, there is no existing work that supports top-k pairs queries based on generic scoring functions. In this chapter, we present a unified approach that supports a broad class of top-k pairs queries including the queries mentioned above. Our proposed approach allows the users to define a local scoring function for each attribute involved in the query and a global scoring function that computes the final score of each pair by combining its scores on different attributes. We propose efficient internal and external memory algorithms and our theoretical analysis shows that the expected performance of the algorithms is optimal when two or less attributes are involved. Our approach does not require any pre-built indexes, is easy to implement and has low memory requirement. We conduct extensive experiments to demonstrate the efficiency of our proposed approach.

Chapter 8

Final Remarks

8.1 Conclusions

In this thesis, we present efficient techniques to answer various proximity based spatial queries under different settings. Chapters 3 to 5 present our research on computing reverse nearest neighbors. We present efficient algorithms to answer moving range queries in Chapter 6. Chapter 7 provides our unified approach to answer top-k pairs queries including k-closest pairs queries and their variants. Below are the details.

In chapter 3, we study the problem of continuous reverse k nearest neighbor monitoring in Euclidean space and in spatial networks. Our proposed approach does not only significantly improve the computation time but also reduces the communication cost for client-server architectures. We also present a thorough theoretical analysis for our Euclidean space RNN algorithm. Furthermore, we show that our algorithms can be extended to handle other variants of RNN queries in Euclidean space and in spatial networks. Experiment results demonstrate an order of magnitude improvement in terms of both the computation time and the communication cost.

In chapter 4, we introduce the concept of influence zone which does not only have applications in target marketing and market analysis but can also be used to answer snapshot and continuous RkNN queries. We present a detailed theoretical analysis to study different aspects of the problem. Extensive experiment results verify the theoretical analysis and demonstrate that influence zone based algorithm outperforms existing algorithms. We also extend our technique to compute influence zone in dimensionality higher than two. We present efficient techniques to update the influence zone as the underlying data set is updated by insertions or deletions of the objects.

In chapter 5, we study the problem of reverse nearest neighbor queries on uncertain data and proposed novel pruning rules that effectively prune the objects that cannot be the RNNs of query. We propose an efficient algorithm and present several optimizations that significantly reduce the overall computation time. Using real data set and synthetic data set, we illustrate the efficiency of our proposed approach. Although we focus on discrete case, the pruning rules we presented can be applied when the uncertain objects are represented by probability density functions.

In chapter 6, we present a safe zone based approach to efficiently monitor distance based range queries in Euclidean space and in road networks. We conduct a rigorous theoretical analysis to study the effectiveness of our safe zone based approach for Euclidean distance based range queries. More specifically, we analyse the probability that a query leaves the safe zone within a time unit and the expected distance it travels before leaving the safe zone. For the queries that satisfy certain constraints, we give an upper bound on the expected number of guard objects. The theoretical results are verified by an extensive experimental study. The experiment results also demonstrate that the proposed approach for Euclidean distance based range queries is close to optimal. We also show that our network distance based algorithm is an order of magnitude faster than a naïve approach.

In chapter 7, we present a unified approach to answer a broad class of top-k pairs query including the k closest pairs queries, the k furthest pairs queries and their variants. We are also first to study the problem of rank-based top-k pairs queries, skyline pairs queries and exclusive top-k pairs queries. The expected performance of the proposed algorithms is optimal when the queries involve two or less attributes. Extensive experiments demonstrate the efficiency of our proposed algorithms.

8.2 Open Problems

In chapter 3, we propose Lazy Updates that computes RNN queries by assigning each object and query a rectangular region called the safe region. We provide a theoretical analysis that shows the relation of the size of the safe region to the computation and communication cost of the system. Ideally, a system should automatically determine a size for the safe regions such that the computation and communication cost is minimized. However, automatic computation of the optimal size of the safe region remains an interesting and challenging open problem. Also, the theoretical analysis for our RNN algorithm for spatial networks is an open problem. Apparently, it is challenging to conduct the analysis even if some quite simple (and probably too strong) assumptions are made (e.g., each edge has equal length, the number of objects on each edge is the same etc.).

Theoretical analysis for the probabilistic reverse nearest neighbors queries is also an open problem. Also, we assumed that the uncertain objects are independent. It will be useful to develop the techniques to answer RNN queries on the data sets that do not satisfy this assumption, i.e., the uncertain objects are correlated.

In chapter 6, we propose a safe zone based approach to continuously monitor the moving range queries. Our technique represents the safe zone by a set of objects called the guard objects. The number of guard objects is important because it affects the overall efficiency as well as the cost of checking whether a query is within its safe zone or not. We provided a theoretical analysis and obtained an upper bound on the number of the guard objects for every query that has the diameter of the safe zone no more than a constant times its expected value. However, the analysis on the expected number of guard objects for the queries that do not satisfy this diameter requirement still remains an open problem.

8.3 Directions for Future Work

In this section, we propose several possible directions for future work.

8.3.1 Influence Zone in Spatial Networks

In Chapter 4, we present the concept of influence zone which is an area such that an object o is the RkNN of q if and only if o is inside the influence zone. We present efficient techniques to compute the influence zone in Euclidean space. A possible future work is to compute the influence zone in spatial networks. As shown in Chapter 4, influence zone based computation of snapshot and continuous RkNN queries in Euclidean space is more efficient than the existing techniques. It will be interesting to see whether the influence zone based RkNN processing technique for spatial networks can outperform the existing techniques. We conjecture that the influence zone based technique to process continuous bichromatic RkNN queries in spatial networks will outperform the existing techniques. This is because once the influence zone has been computed, the technique would require only to monitor the users that enter or leave the influence zone.

8.3.2 Influence Zone Based Communication Efficient Techniques

In Chapter 3, we present Lazy Updates that continuously monitors RkNN queries and does not only improve the computation time but also reduces the communication cost of the system. However, our influence zone based technique to monitor continuous RkNN queries focuses only on reducing the computation time. Recall that, to continuously monitor RkNN queries, our influence zone based algorithm only needs to monitor the users that enter or leave the influence zone. Hence, the algorithm can save the communication cost by requiring the users to report their locations only when they enter the influence zone or when they leave the influence zone. To do this effectively, each user can be assigned a safe region of a simple shape (e.g., a rectangular region) such that as long as it remains inside this region it does not cross the boundary of the influence zone (i.e., it either remains inside the influence zone or it remains outside the influence zone). An effective approach will aim to assign the safe region that i) is of a simple shape so that the user can easily check whether it is inside its safe region or not, and ii) maximizes the time the user remains inside it so that the communication cost is minimized.

8.3.3 Continuous Spatial Queries on Uncertain Data

Most of the existing work on continuous spatial queries assume that the location data is accurate. However, due to the limitation of measuring equipment and delayed data updates, the locations reported by the objects are not always accurate. It is an interesting research direction to develop techniques to continuously monitor various spatial queries by taking into consideration that the locations reported by moving objects and queries are uncertain. Recently, this research direction has gained some attention [TTC+11, BKM+11, PDL11] and it is hoped that, in near future, more techniques will be developed for other spatial queries on continuous uncertain data.

Bibliography

[AKK$^+$09] Elke Achtert, Hans-Peter Kriegel, Peer Kröger, Matthias Renz and Andreas Züfle. Reverse k-nearest neighbor search in dynamic and general metric databases. In *EDBT*, pages 886–897, 2009.

[AP05] Fabrizio Angiulli and Clara Pizzuti. An approximate algorithm for top-k closest pairs join query in large high dimensional data. *Data Knowl. Eng.*, 53(3):263–281, 2005.

[AQ09] Mikhail J. Atallah and Yinian Qi. Computing all skyline probabilities for uncertain data. In *PODS*, pages 279–287, 2009.

[AQY11] Mikhail J. Atallah, Yinian Qi and Hao Yuan. Asymptotically efficient algorithms for skyline probabilities of uncertain data. *ACM Trans. Database Syst.*, 36(2):12, 2011.

[Arg03] Lars Arge. The Buffer Tree: A Technique for Designing Batched External Data Structures. *Algorithmica*, 2003.

[BEK$^+$11] Thomas Bernecker, Tobias Emrich, Hans-Peter Kriegel, Matthias Renz, and Stefan Zankl Andreas Züfle. Efficient Probabilistic Reverse Nearest Neighbor Query Processing on Uncertain Data. In *PVLDB*, 2011.

[Ben75] Jon Louis Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM*, 18(9):509–517, 1975.

[BJKS02] Rimantas Benetis, Christian S. Jensen, Gytis Karciauskas and Simonas Saltenis. Nearest Neighbor and Reverse Nearest Neighbor Queries for Moving Objects. In *IDEAS*, 2002.

[BKM$^+$11] Thomas Bernecker, Hans-Peter Kriegel, Nikos Mamoulis, Matthias Renz and Andreas Züfle. Continuous Inverse Ranking Queries in Uncertain Streams. In *SSDBM*, pages 37–54, 2011.

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider and Bernhard Seeger. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *SIGMOD Conference*, pages 322–331, 1990.

[BKST78] Jon Louis Bentley, H. T. Kung, Mario Schkolnick and Clark D. Thompson. On the Average Number of Maxima in a Set of Vectors and Applications. *J. ACM*, 1978.

[BM72] Rudolf Bayer and Edward M. McCreight. Organization and Maintenance of Large Ordered Indices. *Acta Inf.*, 1:173–189, 1972.

[BO83] Michael Ben-Or. Lower Bounds for Algebraic Computation Trees (Preliminary Report). In *STOC*, 1983.

[Bri02] Thomas Brinkhoff. A Framework for Generating Network-Based Moving Objects. *GeoInformatica*, 2002.

[BSI08] George Beskales, Mohamed Soliman and Ihab Francis Ilyas. Efficient Search for the Top-k Probable Nearest Neighbors in Uncertain Databases. In *VLDB*, 2008.

[CBL$^+$10] Muhammad Aamir Cheema, Ljiljana Brankovic, Xuemin Lin, Wenjie Zhang and Wei Wang. Multi-guarded safe zone: An effective technique to monitor moving circular range queries. In *ICDE*, pages 189–200, 2010.

[CBL$^+$11] Muhammad Aamir Cheema, Ljiljana Brankovic, Xuemin Lin, Wenjie Zhang and Wei Wang. Continuous Monitoring of Distance-Based Range Queries. *IEEE Trans. Knowl. Data Eng.*, 23(8):1182–1199, 2011.

[CC05] Hyung-Ju Cho and Chin-Wan Chung. An Efficient and Scalable Approach to CNN Queries in a Road Network. In *VLDB*, pages 865–876, 2005.

[CCMC08] Reynold Cheng, Jinchuan Chen, Mohamed F. Mokbel and Chi-Yin Chow. Probabilistic Verifiers: Evaluating Constrained Nearest-Neighbor Queries over Uncertain Data. In *ICDE*, pages 973–982, 2008.

[CCX09] Reynold Cheng, Lei Chen 0002, Jinchuan Chen and Xike Xie. Evaluating probability threshold k-nearest-neighbor queries over uncertain data. In *EDBT*, pages 672–683, 2009.

[CHC04] Ying Cai, Kien A. Hua and Guohong Cao. Processing Range-Monitoring Queries on Heterogeneous Mobile Objects. In *Mobile Data Management*, 2004.

[CJT$^+$06] Chee Yong Chan, H. V. Jagadish, Kian-Lee Tan, Anthony K. H. Tung and Zhenjie Zhang. Finding k-dominant skylines in high dimensional space. In *SIGMOD*, 2006.

[CLW$^+$10] Muhammad Aamir Cheema, Xuemin Lin, Wei Wang, Wenjie Zhang and Jian Pei. Probabilistic Reverse Nearest Neighbor Queries on Uncertain Data. *IEEE Trans. Knowl. Data Eng.*, 22(4):550–564, 2010.

[CLW$^+$11] Muhammad Aamir Cheema, Xuemin Lin, Haixun Wang, Jianmin Wang and Wenjie Zhang. A unified approach for computing top-k pairs in multidimensional space. In *ICDE*, pages 1031–1042, 2011.

[CLZ$^+$09] Muhammad Aamir Cheema, Xuemin Lin, Ying Zhang, Wei Wang and Wenjie Zhang. Lazy Updates: An Efficient Technique to Continuously Monitoring Reverse kNN. *PVLDB*, 2(1):1138–1149, 2009.

[CLZZ] Muhammad Aamir Cheema, Xuemin Lin, Wenjie Zhang and Ying Zhang. Efficiently Processing Snapshot and Continuous Reverse k Nearest Neighbors Queries. *The VLDB Journal (currently under review).*

[CLZZ11] Muhammad Aamir Cheema, Xuemin Lin, Wenjie Zhang and Ying Zhang. Influence zone: Efficiently processing reverse k nearest neighbors queries. In *ICDE*, pages 577–588, 2011.

[CMTV00] Antonio Corral, Yannis Manolopoulos, Yannis Theodoridis and Michael Vassilakopoulos. Closest Pair Queries in Spatial Databases. In *SIGMOD*, 2000.

[CPK03] Reynold Cheng, Sunil Prabhakar and Dmitri V. Kalashnikov. Querying Imprecise Data in Moving Object Environments. In *ICDE*, pages 723–725, 2003.

[CSZY09] Zaiben Chen, Heng Tao Shen, Xiaofang Zhou and Jeffrey Xu Yu. Monitoring path nearest neighbor in road networks. In *SIGMOD Conference*, pages 591–602, 2009.

[CXP$^+$04] Reynold Cheng, Yuni Xia, Sunil Prabhakar, Rahul Shah and Jeffrey Scott Vitter. Efficient Indexing Methods for Probabilistic Threshold Queries over Uncertain Data. In *VLDB*, pages 876–887, 2004.

[CZL$^+$11] Muhammad Aamir Cheema, Wenjie Zhang, Xuemin Lin, Ying Zhang and Xuefei Li. Continuous Reverse k Nearest Neighbors Queries in Euclidean Space and in Spatial Networks. *The VLDB Journal*, 2011.

[DYM$^+$05] Xiangyuan Dai, Man Lung Yiu, Nikos Mamoulis, Yufei Tao and Michail Vaitis. Probabilistic Spatial Queries on Existentially Uncertain Data. In *SSTD*, pages 400–417, 2005.

[EKK$^+$10] Tobias Emrich, Hans-Peter Kriegel, Peer Kröger, Matthias Renz and Andreas Züfle. Boosting spatial pruning: on optimal pruning of MBRs. In *SIGMOD Conference*, pages 39–50, 2010.

[Fag99] Ronald Fagin. Combining Fuzzy Information from Multiple Systems. *J. Comput. Syst. Sci.*, 58(1):83–99, 1999.

[FKS03] Ronald Fagin, Ravi Kumar and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *SIGMOD*, 2003.

[FLN03] Ronald Fagin, Amnon Lotem and Moni Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[FNPS79] Ronald Fagin, Jürg Nievergelt, Nicholas Pippenger and H. Raymond Strong. Extendible Hashing - A Fast Access Method for Dynamic Files. *ACM Trans. Database Syst.*, 4(3):315–344, 1979.

[GBK00] Ulrich Güntzer, Wolf-Tilo Balke and Werner Kießling. Optimizing Multi-Feature Queries for Image Databases. In *VLDB*, 2000.

[GG98] Volker Gaede and Oliver Günther. Multidimensional Access Methods. *ACM Comput. Surv.*, 30(2):170–231, 1998.

[GL04] Bugra Gedik and Ling Liu. MobiEyes: Distributed Processing of Continuously Moving Queries on Moving Objects in a Mobile System. In *EDBT*, pages 67–87, 2004.

[GRSY97] Jonathan Goldstein, Raghu Ramakrishnan, Uri Shaft and Jie-Bing Yu. Processing queries by linear constraints. In *PODS*, 1997.

[Gut84] Antonin Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD Conference*, 1984.

[Güt94] Ralf Hartmut Güting. An Introduction to Spatial Database Systems. *VLDB J.*, 3(4):357–399, 1994.

[GWYL04] Bugra Gedik, Kun-Lung Wu, Philip S. Yu and Ling Liu. Motion adaptive indexing for moving continual queries over moving objects. In *CIKM*, 2004.

[Hai94] Eric Haines. *Graphics Gems IV*, Kapitel Point in Polygon Strategies. Academic Press Professional, Cambridge, 1994.

[HCLZ09] Mahady Hasan, Muhammad Aamir Cheema, Xuemin Lin and Ying Zhang. Efficient Construction of Safe Regions for Moving kNN Queries over Dynamic Datasets. In *SSTD*, pages 373–379, 2009.

[HCLZ11] Mahady Hasan, Muhammad Aamir Cheema, Xuemin Lin and Wenjie Zhang. A Unified Algorithm for Continuous Monitoring of Spatial Queries. In *DASFAA (2)*, pages 104–118, 2011.

[HCQL10] Mahady Hasan, Muhammad Aamir Cheema, Wenyu Qu and Xuemin Lin. Efficient Algorithms to Monitor Continuous Constrained Nearest Neighbor Queries. In *DASFAA (1)*, pages 233–249, 2010.

[HS98] Gísli R. Hjaltason and Hanan Samet. Incremental Distance Join Algorithms for Spatial Databases. In *SIGMOD*, 1998.

[HXL05] Haibo Hu, Jianliang Xu and Dik Lun Lee. A Generic Framework for Monitoring Continuous Spatial Queries over Moving Objects. In *SIGMOD Conference*, pages 479–490, 2005.

[IK95] Christian Icking and Rolf Klein. Searching for the Kernel of a Polygon - A Competitive Strategy. In *Symposium on Computational Geometry*, pages 258–266, 1995.

[ISS03] Glenn S. Iwerks, Hanan Samet and Kenneth P. Smith. Continuous K-Nearest Neighbor Queries for Continuously Moving Points with Updates. In *VLDB*, pages 512–523, 2003.

[JKPT03] Christian S. Jensen, Jan Kolárvr, Torben Bach Pedersen and Igor Timko. Nearest neighbor queries in road networks. In *GIS*, pages 1–8, 2003.

[KF93] Ibrahim Kamel and Christos Faloutsos. On Packing R-trees. In *CIKM*, 1993.

[KKPR06] Hans-Peter Kriegel, Peter Kunath, Martin Pfeifle and Matthias Renz. Probabilistic Similarity Join on Uncertain Data. In *DASFAA*, pages 295–309, 2006.

[KKR07] Hans-Peter Kriegel, Peter Kunath and Matthias Renz. Probabilistic Nearest-Neighbor Query on Uncertain Objects. In *DASFAA*, pages 337–348, 2007.

[KKR08] Hans-Peter Kriegel, Peer Kröger and Matthias Renz. Continuous proximity monitoring in road networks. In *GIS*, page 12, 2008.

[KM00] Flip Korn and S. Muthukrishnan. Influence Sets Based on Reverse Nearest Neighbor Queries. In *SIGMOD*, 2000.

[KMS$^+$07] James M. Kang, Mohamed F. Mokbel, Shashi Shekhar, Tian Xia and Donghui Zhang. Continuous Evaluation of Monochromatic and Bichromatic Reverse Nearest Neighbors. In *ICDE*, 2007.

[KS04a] M. Kolahdouzan and Cyrus Shahabi. Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases. In *VLDB*, pages 840–851, 2004.

[KS04b] Mohammad R. Kolahdouzan and Cyrus Shahabi. Continuous K-Nearest Neighbor Queries in Spatial Network Databases. In *STDBM*, pages 33–40, 2004.

[KT06] Axel Küpper and Georg Treu. Efficient proximity and separation detection among mobile targets for supporting location-based community services. *Mobile Computing and Communications Review*, 10(3):1–12, 2006.

[L09] Xiang Lian and Lei Chen 0002. Efficient processing of probabilistic reverse nearest neighbor queries over uncertain data. *VLDB J.*, 18(3):787–808, 2009.

[LDH06] Fuyu Liu, Tai T. Do and Kien A. Hua. Dynamic Range Query in Spatial Network Environments. In *DEXA*, pages 254–265, 2006.

[LLL$^+$10] Guohui Li, Yanhong Li, Jianjun Li, LihChyun Shu and Fumin Yang. Continuous reverse k nearest neighbor monitoring on moving objects in road networks. *Inf. Syst.*, 35(8):860–883, 2010.

[LNY03] King-Ip Lin, Michael Nolen and Congjun Yang. Applying Bulk Insertion Techniques for Dynamic Reverse Nearest Neighbor Problems. *IDEAS*, 2003.

[LPM02] Iosif Lazaridis, Kriengkrai Porkaew and Sharad Mehrotra. Dynamic Queries over Mobile Objects. In *EDBT*, pages 269–286, 2002.

[LZZC11] Xuemin Lin, Ying Zhang, Wenjie Zhang and Muhammad Aamir Cheema. Stochastic skyline operator. In *ICDE*, pages 721–732, 2011.

[MCA06] Mohamed F. Mokbel, Chi-Yin Chow and Walid G. Aref. The New Casper: Query Processing for Location Services without Compromising Privacy. In *VLDB*, pages 763–774, 2006.

[MHP05] Kyriakos Mouratidis, Marios Hadjieleftheriou and Dimitris Papadias. Conceptual Partitioning: An Efficient Method for Continuous Nearest Neighbor Monitoring. In *SIGMOD*, 2005.

[MPBT05] Kyriakos Mouratidis, Dimitris Papadias, Spiridon Bakiras and Yufei Tao. A Threshold-Based Algorithm for Continuous Monitoring of k Nearest Neighbors. *TKDE*, pages 1451–1464, 2005.

[MXA04] Mohamed F. Mokbel, Xiaopeng Xiong and Walid G. Aref. SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases. In *SIGMOD Conference*, pages 623–634, 2004.

[MYCC07] Nikos Mamoulis, Man Lung Yiu, Kit Hung Cheng and David W. Cheung. Efficient top- aggregation of ranked inputs. *ACM Trans. Database Syst.*, 32(3):19, 2007.

[MYPM06] Kyriakos Mouratidis, Man Lung Yiu, Dimitris Papadias and Nikos Mamoulis. Continuous Nearest Neighbor Monitoring in Road Networks. In *VLDB*, pages 43–54, 2006.

[NR99] Surya Nepal and M. V. Ramakrishna. Query Processing Issues in Image (Multimedia) Databases. In *ICDE*, 1999.

[NZTK08] Sarana Nutanong, Rui Zhang, Egemen Tanin and Lars Kulik. The V*-Diagram: a query-dependent approach to moving KNN queries. *PVLDB*, 1(1):1095–1106, 2008.

[OBSC99] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara and Sung Nok Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley, 1999.

[PBGK10] Michalis Potamias, Francesco Bonchi, Aristides Gionis and George Kollios. k-Nearest Neighbors in Uncertain Graphs. *PVLDB*, 3(1):997–1008, 2010.

[PDL11] Liping Peng, Yanlei Diao and Anna Liu. Optimizing Probabilistic Query Processing on Continuous Uncertain Data. *PVLDB*, 4(11):1169–1180, 2011.

[PJLY07] Jian Pei, Bin Jiang, Xuemin Lin and Yidong Yuan. Probabilistic Skylines on Uncertain Data. In *VLDB*, pages 15–26, 2007.

[PS85] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry An Introduction*. Springer, 1985.

[PTFS05] Dimitris Papadias, Yufei Tao, Greg Fu and Bernhard Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82, 2005.

[PXK$^+$02] Sunil Prabhakar, Yuni Xia, Dmitri V. Kalashnikov, Walid G. Aref and Susanne E. Hambrusch. Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects. *IEEE Trans. Computers*, 51(10):1124–1140, 2002.

[PZMT03] Dimitris Papadias, Jun Zhang, Nikos Mamoulis and Yufei Tao. Query Processing in Spatial Network Databases. In *VLDB*, pages 802–813, 2003.

[QTJ$^+$08] Shaojie Qiao, Changjie Tang, Huidong Jin, Shucheng Dai and Xingshu Chen. Constrained k-closest pairs query processing based on growing window in crime databases. In *ISI*, 2008.

[SAA00] Ioana Stanoi, Divyakant Agrawal and Amr El Abbadi. Reverse Nearest Neighbor Queries for Dynamic Databases. In *ACM SIGMOD Workshop*, 2000.

[SET09] Maytham Safar, Dariush Ebrahimi and David Taniar. Voronoi-based reverse nearest neighbor query processing on spatial networks. *Multimedia Syst.*, 15(5):295–308, 2009.

[SFT03] Amit Singh, Hakan Ferhatosmanoglu and Ali Saman Tosun. High dimensional reverse nearest neighbor queries. In *CIKM*, 2003.

[SJLS08] Huan-Liang Sun, Chao Jiang, Jun-Ling Liu and Limei Sun. Continuous Reverse Nearest Neighbor Queries on Moving Objects in Road Networks. In *WAIM*, pages 238–245, 2008.

[SKS02] Cyrus Shahabi, Mohammad R. Kolahdouzan and Mehdi Sharifzadeh. A road network embedding technique for k-nearest neighbor search in moving object databases. In *ACM-GIS*, pages 94–10, 2002.

[Smi97] Michiel Smid. Closest-Point Problems in Computational Geometry. In *Handbook on Computational Geometry, published by Elsevier Science*, 1997.

[SPP$^+$08] Dragan Stojanovic, Apostolos N. Papadopoulos, Bratislav Predic, Slobodanka Djordjevic-Kajan and Alexandros Nanopoulos. Continuous range monitoring of mobile objects in road networks. *Data Knowl. Eng.*, 64(1):77–100, 2008.

[SR01] Zhexuan Song and Nick Roussopoulos. K-Nearest Neighbor Search for Moving Query Point. In *SSTD*, pages 79–96, 2001.

[SRAA01] Ioana Stanoi, Mirek Riedewald, Divyakant Agrawal and Amr El Abbadi. Discovery of Influence Sets in Frequently Updated Databases. In *VLDB*, 2001.

[SSH86] Michael Stonebraker, Timos K. Sellis and Eric N. Hanson. An Analysis of Rule Indexing Implementations in Data Base Systems. In *Expert Database Conf.*, pages 465–476, 1986.

[SY03] Shashi Shekhar and Jin Soung Yoo. Processing in-route nearest neighbor queries: a comparison of alternative approaches. In *GIS*, pages 9–16, 2003.

[SZS03] Jing Shan, Donghui Zhang and Betty Salzberg. On Spatial-Range Closest-Pair Query. In *SSTD*, pages 252–269, 2003.

[TCX$^+$05] Yufei Tao, Reynold Cheng, Xiaokui Xiao, Wang Kay Ngai, Ben Kao and Sunil Prabhakar. Indexing Multi-Dimensional Uncertain Data with Arbitrary Probability Density Functions. In *VLDB*, pages 922–933, 2005.

[TP02] Yufei Tao and Dimitris Papadias. Time-parameterized queries in spatio-temporal databases. In *SIGMOD Conference*, pages 334–345, 2002.

[TPL04] Yufei Tao, Dimitris Papadias and Xiang Lian. Reverse kNN search in arbitrary dimensionality. In *VLDB*, 2004.

[TPS02] Yufei Tao, Dimitris Papadias and Qiongmao Shen. Continuous Nearest Neighbor Search. In *VLDB*, pages 287–298, 2002.

[TSS00] Yannis Theodoridis, Emmanuel Stefanakis and Timos K. Sellis. Efficient Cost Models for Spatial Queries Using R-Trees. *IEEE Trans. Knowl. Data Eng.*, 2000.

[TTC$^+$11] Goce Trajcevski, Roberto Tamassia, Isabel F. Cruz, Peter Scheuermann, David Hartglass and Christopher Zamierowski. Ranking continuous nearest neighbors for uncertain trajectories. *VLDB J.*, 20(5):767–791, 2011.

[TTS09] Quoc Thai Tran, David Taniar and Maytham Safar. Reverse k Nearest Neighbor and Reverse Farthest Neighbor Search on Spatial Networks. *T. Large-Scale Data- and Knowledge-Centered Systems*, 1:353–372, 2009.

[TYM06] Yufei Tao, Man Lung Yiu and Nikos Mamoulis. Reverse Nearest Neighbor Search in Metric Spaces. *TKDE*, 18(9), 2006.

[UMY07] Leong Hou U, Nikos Mamoulis and Man Lung Yiu. Continuous Monitoring of Exclusive Closest Pairs. In *SSTD*, 2007.

[Vit01] Jeffrey Scott Vitter. External Memory Algorithms and Data Structures: Dealing with Massive Data. *ACM Computing Surveys*, 33:2001, 2001.

[WCY06] Kun-Lung Wu, Shyh-Kwei Chen and Philip S. Yu. Incremental Processing of Continual Range Queries over Moving Objects. *IEEE Trans. Knowl. Data Eng.*, 18(11):1560–1575, 2006.

[Wid05] Jennifer Widom. Trio: A System for Integrated Management of Data, Accuracy, and Lineage. In *CIDR*, pages 262–276, 2005.

[WW06] Xiaoyuan Wang and Wei Wang. Continuous Expansion: Efficient Processing of Continuous Range Monitoring in Mobile Environments. In *DASFAA*, pages 890–899, 2006.

[WYCT08a] Wei Wu, Fei Yang, Chee Yong Chan and Kian-Lee Tan. Continuous Reverse k-Nearest-Neighbor Monitoring. In *MDM*, 2008.

[WYCT08b] Wei Wu, Fei Yang, Chee Yong Chan and Kian-Lee Tan. Finch: Evaluating Reverse k-Nearest-Neighbor Queries on Location Data. In *VLDB*, 2008.

[WZ08] Haojun Wang and Roger Zimmermann. Snapshot location-based query processing on moving objects in road networks. In *GIS*, page 50, 2008.

[WZK06] Haojun Wang, Roger Zimmermann and Wei-Shinn Ku. Distributed Continuous Range Query Processing on Moving Objects. In *DEXA*, pages 655–665, 2006.

[XMA05] Xiaopeng Xiong, Mohamed F. Mokbel and Walid G. Aref. SEA-CNN: Scalable Processing of Continuous K-Nearest Neighbor Queries in Spatio-temporal Databases. In *ICDE*, pages 643–654, 2005.

[XZ06] Tian Xia and Donghui Zhang. Continuous Reverse Nearest Neighbor Monitoring. In *ICDE*, page 77, 2006.

[YL01] Congjun Yang and King-Ip Lin. An Index Structure for Efficient Reverse Nearest Neighbor Queries. In *ICDE*, 2001.

[YL02] Congjun Yang and King-Ip Lin. An index structure for improving nearest closest pairs and related join queries in spatial databases. In *IDEAS*, 2002.

[YM07] Man Lung Yiu and Nikos Mamoulis. Reverse Nearest Neighbors Search in Ad Hoc Subspaces. *TKDE*, 19(3):412–426, 2007.

[YPK05] Xiaohui Yu, Ken Q. Pu and Nick Koudas. Monitoring K-Nearest Neighbor Queries Over Moving Objects. In *ICDE*, 2005.

[YPMT05] Man Lung Yiu, Dimitris Papadias, Nikos Mamoulis and Yufei Tao. Reverse Nearest Neighbors in Large Graphs. In *ICDE*, 2005.

[ZL01] Baihua Zheng and Dik Lun Lee. Semantic Caching in Location-Dependent Query Processing. In *SSTD*, pages 97–116, 2001.

[ZLZ$^+$09] Wenjie Zhang, Xuemin Lin, Ying Zhang, Wei Wang and Jeffrey Xu Yu. Probabilistic Skyline Operator over Sliding Windows. In *ICDE*, pages 1060–1071, 2009.

[ZZL$^+$11] Ying Zhang, Wenjie Zhang, Xuemin Lin, Bin Jiang and Jian Pei. Ranking uncertain sky: The probabilistic top-k skyline operator. *Inf. Syst.*, 36(5):898–915, 2011.

[ZZP$^+$03] Jun Zhang, Manli Zhu, Dimitris Papadias, Yufei Tao and Dik Lun Lee. Location-based Spatial Queries. In *SIGMOD Conference*, pages 443–454, 2003.

Appendix A

Proofs

A.1 Related Glossary

Antipodal Corners: Let C be a corner of rectangle $R1$ and C' be a corner in $R2$, the two corners are called *antipodal corners* if for every dimension i where $C[i] = R1_L[i]$ then $C'[i] = R2_H[i]$ and for every dimension j where $C[j] = R1_H[j]$ then $C'[j] = R2_L[j]$. Fig A.1 shows two rectangles $R1$ and $R2$. The corners D and O are antipodal corners. Similarly, other pairs of antipodal corners are (B, M), (C, N) and (A, P).

Antipodal half space: A half space that is defined by the bisector between two antipodal corners is called *antipodal half space*. Fig A.1 shows two antipodal half spaces $H_{M:B}$ and $H_{P:A}$.

Normalized half space: Let M and B be two points in hyper-rectangles R and Q, respectively. The normalized half space $H'_{M:B}$ is a space defined by a bisector between M and B that passes through a point c such that $c[i] = (Q_L[i] + R_L[i])/2$ for all dimensions i for which $B[i] > M[i]$ and $c[j] = (Q_H[i] + R_H[j])/2$ for all dimensions j for which $B[j] \le M[j]$. Fig A.1 shows two normalized (antipodal) half spaces $H'_{M:B}$ and $H'_{P:A}$. The point c for each half space is also shown. The inequalities (A.1) and (A.2) define the half space $H_{M:B}$ and its normalized half space $H'_{M:B}$, respectively.

$$\sum_{i=1}^{d}(B[i] - M[i]) \cdot x[i] < \sum_{i=1}^{d} \frac{(B[i] - M[i])(B[i] + M[i])}{2} \tag{A.1}$$

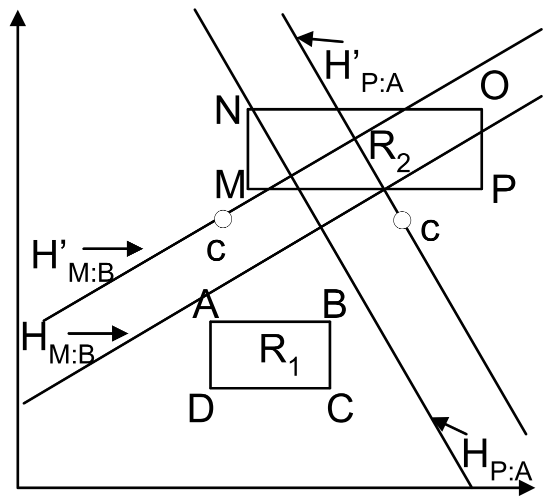


Figure A.1: Antipodal corners and half spaces

$$\sum_{i=1}^{d}(B[i]-M[i])\cdot x[i] < \sum_{i=1}^{d}(B[i]-M[i]) \times \left\{ \begin{array}{ll} \dfrac{(Q_L[i]+R_L[i])}{2} & \text{if } B[i] > M[i]) \\ \dfrac{(Q_H[i]+R_H[i])}{2} & \text{otherwise} \end{array} \right\} \tag{A.2}$$

Note that the right hand side of the inequality (A.1) can never be smaller than the right hand side of inequality (A.2) because M and B both lie in hyper-rectangles R and Q, respectively. For this reason $H'_{M:B} \subseteq H_{M:B}$.

Set of More Expressive Half Spaces: A set of half spaces $S_1 = \{H_{i:q}, ..., H_{n:q}\}$ is more expressive than any other half space $H_{j:q}$ if it holds that $\cap_{x=i}^{n} H_{x:q} \subseteq H_{j:q}$. Note that if S_1 is a set of more expressive half spaces then $\cap_{x=i}^{n} H_{x:q} \cap H_{j:q} = \cap_{x=i}^{n} H_{x:q}$. For example, the set of half spaces $\{H_{M:q}, H_{N:q}\}$ in Fig. A.2 is more expressive than the half space $H_{L:q}$ and the shaded area is $H_{M:q} \cap H_{N:q} \cap H_{L:q} = H_{M:q} \cap H_{N:q}$.

A.2 Proofs

Lemma A.2.1 *Let there be two subspaces SP_1 and SP_2;*

$$SP_1 \Rightarrow y < Ax + B \tag{A.3}$$

$$SP_2 \Rightarrow y < Cx + D \tag{A.4}$$

where x and y are variables and A, B, C and D are constants. Both the subspaces intersect each other at $x = I_x = \frac{D-B}{A-C}$. If the whole space is partitioned into two partitions Pn_1 and Pn_2 such that Pn_1 contains all the points for which $x \geq I_x$ and Pn_2 contains all the points where $x \leq I_x$. Then we can say;

$$\left\{\begin{array}{ll} \left\{\begin{array}{c} SP_1 \subseteq SP_2; \quad in\ Pn_1 \\ AND \\ SP_2 \subseteq SP_1; \quad in\ Pn_2 \end{array}\right\} & if \quad C > A \\ \left\{\begin{array}{c} SP_2 \subseteq SP_1; \quad in\ Pn_1 \\ AND \\ SP_1 \subseteq SP_2; \quad in\ Pn_2 \end{array}\right\} & otherwise \end{array}\right\}$$

Proof We prove the case when $C > A$ and the proof of the other case is similar. Note that for $x = I_x$, the right hand sides of both the inequalities (A.3) and (A.4) would be equal and for $x > I_x$ the right hand side of the inequality (A.4) is greater than right hand side of inequality (A.3) because $C > A$. This means every point that lies in Pn_1 and satisfies inequality (A.3) would also satisfy the inequality (A.4). Hence $SP_1 \subseteq SP_2$ in space where $x \geq I_x$. Similarly, it can be proved that $SP_2 \subseteq SP_1$ in space where $x \leq I_x$. Also the proof for the case when $C \leq A$ is similar. ∎

Lemma A.2.2 *Let there be three half spaces SP_1, SP_2 and SP_3 defined by the following inequalities;*

$$SP_1 \Rightarrow y < Ax + B \tag{A.5}$$

$$SP_2 \Rightarrow y < Cx + D \tag{A.6}$$

$$SP_3 \Rightarrow y < Ex + F \tag{A.7}$$

where x and y are variables and A, B, C, D, E and F are constants. The set of half spaces $\{SP_1, SP_2\}$ is always more expressive[1] *than SP_3 if both of the following are true;*

1. $A > E > C$
2. $\frac{F-B}{A-E} \geq \frac{D-F}{E-C}$

Proof Since $A > E > C$, we can obtain from Lemma A.2.1;

$$SP_2 \subseteq SP_3; \quad if\ x \geq \frac{D-F}{E-C} \tag{A.8}$$

$$SP_3 \subseteq SP_2; \quad if\ x \leq \frac{D-F}{E-C} \tag{A.9}$$

$$SP_3 \subseteq SP_1; \quad if\ x \geq \frac{F-B}{A-E} \tag{A.10}$$

$$SP_1 \subseteq SP_3; \quad if\ x \leq \frac{F-B}{A-E} \tag{A.11}$$

Since $\frac{F-B}{A-E} \geq \frac{D-F}{E-C}$, we obtain by joining the inequalities (A.8) and (A.10);

$$SP_2 \subseteq SP_3 \subseteq SP_1; \quad if\ x \geq \frac{F-B}{A-E} \tag{A.12}$$

From inequalities (A.12) and (A.11), it can be noted that in the whole space SP_3 is either a superset of SP_1 or SP_2. Hence $SP_1 \cap SP_2 \subseteq SP_3$. ■

[1]The set of more expressive half spaces is defined in Glossary (Section A.1).

Lemma A.2.3 *Let M and N be two points in d-dimensional space such that $M[i] = N[i]$ for all except one dimension j. Let q be a query point and MN be the line joining the points M and N. The set of half spaces $\{H_{M:q}, H_{N:q}\}$ is more expressive than any $H_{L:q}$ where L is any point on the line segment MN. Fig. A.2 shows the line and half spaces in 2d space.*

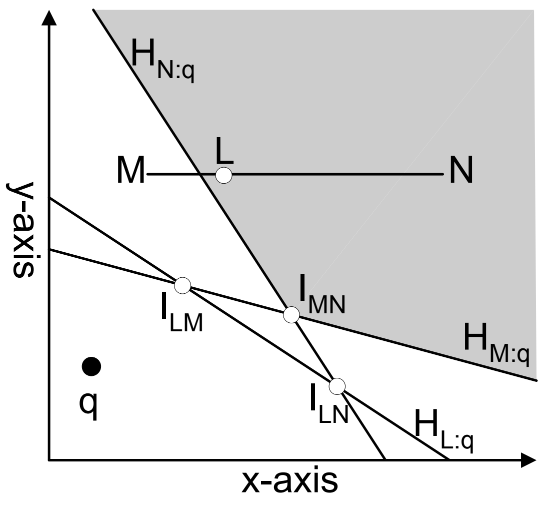


Figure A.2: Lemma A.2.3 in 2d-space

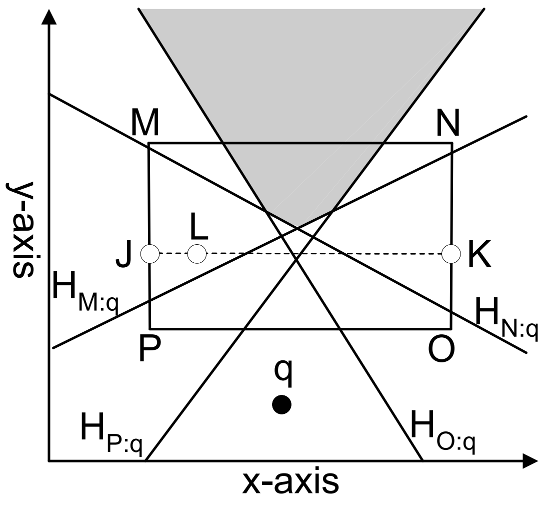


Figure A.3: Lemma A.2.4 in 2d-space

Proof The half-subspace $H_{N:q}$ and $H_{M:q}$ are defined by inequality (A.13) and inequality (A.14), respectively;

$$\sum_{i=1, i\neq j}^{d} (q[i] - N[i]) \cdot x[i] < (N[j] - q[j]) \cdot x[j] + \sum_{i=1}^{d} (q[i]^2 - N[i]^2)/2 \tag{A.13}$$

$$\sum_{i=1, i\neq j}^{d} (q[i] - M[i]) \cdot x[i] < (M[j] - q[j]) \cdot x[j] + \sum_{i=1}^{d} (q[i]^2 - M[i]^2)/2 \tag{A.14}$$

Let $A = (N[j] - q[j])$, $B = \sum_{i=1}^{d}(q[i]^2 - N[i]^2)/2$, $C = (M[j] - q[j])$ and $D = \sum_{i=1}^{d}(q[i]^2 - M[i]^2)/2$ be constants and $y = \sum_{i=1, i\neq j}^{d}(q[i] - M[i]) \cdot x[i]$ be a variable. Note that $M[i] = N[i]$ for all except j^{th} dimension, so we can write inequalities (A.13) and (A.14) as;

$$H_{N:q} \Rightarrow y < A \cdot x[j] + B \tag{A.15}$$

$$H_{M:q} \Rightarrow y < C \cdot x[j] + D \tag{A.16}$$

For any point L on the line MN, let $E = (L[j] - q[j])$ and $F = \sum_{i=1}^{d}(q[i]^2 - L[i]^2)/2$ be a constant. Then $H_{L:q}$ is represented by the inequality (A.17);

$$H_{L:q} \Rightarrow y < E \cdot x[j] + F \tag{A.17}$$

Without loss of generality, if we assume $M < L < N$ then $A > E > C$. Since $M[i] = N[i] = L[i]$ for all except j^{th} dimension, we calculate $\frac{F-B}{A-E}$ and $\frac{D-F}{E-C}$ which are $(N[j] + L[j])/2$ and $(M[j] + L[j])/2$, respectively. Since $\frac{F-B}{A-E} > \frac{D-F}{E-C}$, it is proved from Lemma A.2.2 that the set of half spaces $\{H_{M:q}, H_{N:q}\}$ is more expressive than any $H_{L:q}$. ∎

Lemma A.2.4 *Let q be a query point, R be a hyper-rectangle in d-dimensional space and $\{C_1, C_2, ..., C_{2^d}\}$ be its corners. The set of half spaces $\{H_{C_1:q}, H_{C_2:q}, ..., H_{C_{2^d}:q}\}$ is more expressive than every other half space $H_{L:q}$ where L is any point in the hyper-rectangle R.*

Proof We present the proof for a $2d$-rectangle and it can be extended to prove the Lemma for high-dimensional hyper-rectangles. In Fig. A.3, a rectangle has been shown with four corners M, N, O and P. Note that for every point L in rectangle there exist two points J and K on the boundary of rectangle such that $\{H_{J:q}, H_{K:q}\}$ is more expressive than $H_{L:q}$ (Lemma A.2.3). For the same reasoning, note that $\{H_{N:q}, H_{O:q}\}$ is more expressive than $H_{K:q}$ and $\{H_{M:q}, H_{P:q}\}$ is more expressive than $H_{J:q}$. Hence $\{H_{M:q}, H_{N:q}, H_{O:q}, H_{P:q}\}$ is a set of more expressive half spaces than every half space $H_{L:q}$. It is easy to see that this reasoning can be extended to prove the Lemma for hyper-rectangles in higher-dimensions. ∎

Lemma A.2.5 *Let there be two d dimensional hyper-rectangles Q and R. The set of normalized half spaces $\{H'_{C_1:C'_1}, ..., H'_{C_{2d}:C'_{2d}}\}$ is more expressive than any half space $H_{M:N}$ where C_i is i^{th} corner of R and C'_i is its antipodal corner in Q.*

Proof Let M be any point in hyper-rectangle R and N be any point in hyper-rectangle Q. If we prove that the set of normalized half spaces $\{H'_{C_1:C'_1}, ..., H'_{C_{2d}:C'_{2d}}\}$ is more expressive than any normalized half space $H'_{M:N}$, we can say that it is more expressive than the half space $H_{M:N}$ because $H'_{M:N} \subseteq H_{M:N}$ by the definition of normalized half spaces.

Unless the two points M and N are antipodal corners, it holds true that there exist two points Y and Z in R and Q, respectively, such that for all dimensions i except j, $Y[i] = M[i]$ and $Z[i] = N[i]$ and for dimension j at least one of the following two holds true;

Case 1: $(Y[j] = R_H[j]) > M[j]$ and $(Z[j] = Q_H[j]) > N[j]$

Case 2: $(Y[j] = R_L[j]) < M[j]$ and $(Z[j] = Q_L[j]) < N[j]$

We present the proof for case 1 and the proof for case 2 is similar. Let A, B, C, D, E, F and G be constants and y be variable defined as;

$$y = \sum_{i=1, i\neq j}^{d} (N[i] - M[i]) \cdot x[i]$$

$$A = Y[j] - N[j] = R_H[j] - N[j]$$

$$C = M[j] - Z[j] = M[j] - Q_H[j]$$

$$E = M[j] - N[j]$$

$$G = \sum_{i=1, i\neq j}^{d} \frac{N[i] - M[i]}{2} \times \left\{ \begin{array}{l} (Q_L[i] + R_L[i]);\ \text{if } N[i] > M[i] \\ (Q_H[i] + R_H[i]);\ \text{otherwise} \end{array} \right\}$$

$$B = G + \frac{N[j] - Y[j]}{2} \times \begin{Bmatrix} (Q_L[j] + R_L[j]); \text{ if } N[j] > Y[j] \\ (Q_H[j] + R_H[j]); \text{ otherwise} \end{Bmatrix}$$

$$D = G + \frac{Z[j] - M[j]}{2} \times \begin{Bmatrix} (Q_L[j] + R_L[j]); \text{ if } Z[j] > M[j] \\ (Q_H[j] + R_H[j]); \text{ otherwise} \end{Bmatrix}$$

$$F = G + \frac{N[j] - M[j]}{2} \times \begin{Bmatrix} (Q_L[j] + R_L[j]); \text{ if } N[j] > M[j] \\ (Q_H[j] + R_H[j]); \text{ otherwise} \end{Bmatrix}$$

The normalized half spaces $H'_{Y:N}, H'_{M:Z}$ and $H'_{M:N}$ are defined by the following inequalities.

$$H'_{Y:N} \Rightarrow y < A \cdot x[j] + B \tag{A.18}$$

$$H'_{M:Z} \Rightarrow y < C \cdot x[j] + D \tag{A.19}$$

$$H'_{M:N} \Rightarrow y < E \cdot x[j] + F \tag{A.20}$$

According to the Lemma A.2.2, if $A > E > C$ and $\frac{F-B}{A-E} \geq \frac{D-F}{E-C}$ then the set of normalized half spaces $\{H'_{Y:N}, H'_{M:Z}\}$ is more expressive than the normalized half space $H'_{M:N}$. It is easy to observe that $A > E > C$ now we compute $\frac{F-B}{A-E}$ and $\frac{D-F}{E-C}$. There are two possibilities.

Possibility 1: $N[j] \leq M[j]$; In this case $N[j]$ is always less than $Y[j]$ and $\frac{F-B}{A-E} = \frac{(Q_H[i]+R_H[i])}{2}$. On the other hand $Z[j]$ might be greater, lesser or equal to $M[j]$. To maximize $\frac{D-F}{E-C}$, we assume that $Z[j] > M[j]$ and compute $\frac{D-F}{E-C} = \frac{(Q_L[i]+R_L[i])}{2}$. Hence $\frac{F-B}{A-E} > \frac{D-F}{E-C}$.

Possibility 2:$N[j] > M[j]$; In this case $Z[j]$ is always greater than $M[j]$. We can compute that $\frac{D-F}{E-C} = \frac{(Q_L[i]+R_L[i])}{2}$. On the other hand $N[j]$ might be greater, lesser or equal to $Y[j]$. To minimize $\frac{F-B}{A-E}$, we assume that $N[j] > Y[j]$ and compute $\frac{F-B}{A-E} = \frac{(Q_L[i]+R_L[i])}{2}$. Hence $\frac{F-B}{A-E} \geq \frac{D-F}{E-C}$.

We have proved that the set of normalized half spaces $\{H'_{M:Z}, H'_{Y:N}\}$ is more expressive than the normalized half space $H'_{M:N}$. It can be found that for any such $H'_{M:Z}$ (or $H'_{Y:N}$), there exists a set of normalized half spaces that is more expressive unless M and Z (or Y and N) are two antipodal corners. Hence the set of antipodal normalized half spaces $\{H'_{C_1:C'_1}, ..., H'_{C_{2^d}:C'_{2^d}}\}$ is more expressive than any other normalized half space $H'_{M:N}$ where M and N are the points in hyper-rectangle R and Q, respectively. Since $H'_{M:N} \subseteq H_{M:N}$, we can say that the set $\{H'_{C_1:C'_1}, ..., H'_{C_{2^d}:C'_{2^d}}\}$ is more expressive than any half space $H_{M:N}$. This completes the proof. ∎

Lemma A.2.6 *Let Q and R be two hyper-rectangles in d dimensional space such that for every dimension i, either $R_H[i] \leq Q_L[i]$ or $Q_H[i] \leq R_L[i]$ and for at least one dimension j either $R_H[j] < Q_L[j]$ or $Q_H[j] < R_L[j]$ (i.e; there exists a dominance relationship such that R is dominated by Q). Let F_p and p be two points such that $p > (F_p[i] = (Q_H[i]+R_H[i])/2)$ for any dimension i for which $Q_H[i] \leq R_L[i]$ and $p < (F_p[j] = (Q_L[j]+R_L[j])/2)$ for any dimension j for which $R_H[j] \leq Q_L[j]$ (i.e; p is dominated by F_p in the same way as R is dominated by Q). Then we can say $maxdist(p, R) > mindist(p, Q)$.*

Proof We can prove the lemma by showing that the point p lies in every normalized half space $H'_{M:N}$ where M is a point in R and N is a point in Q. The normalized half space can be defined as;

$$\begin{aligned}&\sum_{i=1}^{d}(N[i] - M[i]) \cdot x[i] < \\ &\sum_{i=1}^{d}(N[i] - M[i]) \times \left\{ \begin{array}{ll} \dfrac{(Q_L[i] + R_L[i])}{2} & \text{if } N[i] > M[i]) \\ \dfrac{(Q_H[i] + R_H[i])}{2} & \text{otherwise} \end{array} \right\}\end{aligned} \tag{A.21}$$

We evaluate the left hand side of the inequality (A.21) w.r.t F_p (e.g; $x[i] = F_p[i]$);

$$\sum_{i=1}^{d}(N[i] - M[i]) \times \left\{ \begin{array}{ll} \dfrac{Q_L[i] + R_L[i])}{2} & \text{if } Q_L[i] \geq R_H[i] \\ \dfrac{Q_H[i] + R_H[i])}{2} & \text{if } R_L[i] \geq Q_H[i] \end{array} \right\} \tag{A.22}$$

It can be observed that the value in (A.22) is always equal to the RHS of the inequality (A.21) because M is a point in R and N is a point in Q. So for any dimension i where $Q_L[i] \geq R_H[i]$, $N[i] - M[i]$ is always positive. Similarly, for any dimension j for which $R_L[i] \geq Q_H[i]$, $N[i] - M[i]$ is always negative.

Furthermore, it can be noted by the definition of the point p that the LHS of the inequality (A.21) when evaluated w.r.t p is always less than what we obtained in (A.22). Hence p lies in every normalized half space $H'_{M:N}$. ∎

Index